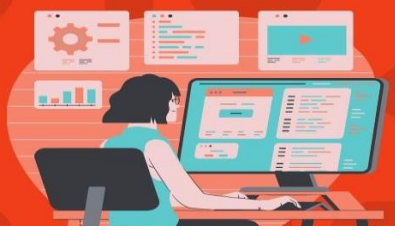


LENGUAJES Y AUTÓMATAS II

ME XENIA PADILLA MADRID

ACTIVIDAD DE LABORATORIO	**12**
NOMBRE DE LA ACTIVIDAD	IF Y SUS VARIACIONES JUNTO CON SCOPPE DE VARIABLES
NOMBRE ESTUDIANTE	Itsai Zempoaltecatl Mejia
GRUPO	6SA
FECHA	27/abril/2022

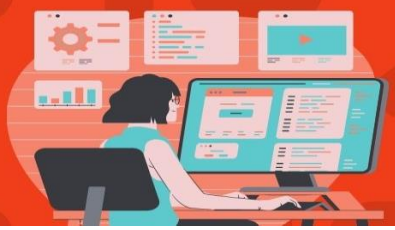
```
public static HashMap<String, Object> memory = new HashMap<~>();
public static HashMap<String, Object> tempMemory = new HashMap<>();
private boolean joinIfElse = false;
private boolean bodyInScope=false;
```

```
@Override
public Object visitSentenciaElse(OpmezParser.SentenciaElseContext ctx) {
    Object result = null;
    joinIfElse = true;
    try{
        result=visit(ctx.body());
    }catch(Exception e){
        errors++;
        ps.println("Algo fallo en: else");
    }

    tempMemory.clear();
    return result;
}
```

```
@Override
public Object visitCuerpoScope(OpmezParser.CuerpoScopeContext ctx) {
    for (int i = 0; i < ctx.instructions().size(); i++) {
        visit(ctx.instructions(i));
        bodyInScope = true;
    }
    bodyInScope=false;
    return null;
}
```

Para el scoppe de variables declare 2 HashMaps y 2 variables las cuales que me ayudaran a resolver esto.
memory: es la memoria de variables globales, o sea que si son declaradas en el cuerpo principal del programa pueden ser utilizadas en if-else-elif que estén después.
tempMemory: es la memoria para las variables temporales que sean declaradas dentro de if-else-elif.
joinIfElse: es la variable encargada de detectar que el código se este ejecutando dentro de if-else-elif para así hacer las validaciones necesarias de las variables, en cuanto la ejecución detecta que hay un if-else-elif pasa a ser true.
bodyInScope: esta variable esta encargada de validar la lectura de cada sentencia del cuerpo de if-else-elif.



LENGUAJES Y AUTÓMATAS II

ME XENIA PADILLA MADRID

```java
@Override
public Object visitDeclaracion(OpmezParser.DeclaracionContext ctx) {
    String id = ctx.ID().getText();

    if(joinIfElse || bodyInScope){
        if(memory.containsKey(id) && tempMemory.containsKey(id)){
            errorDeclaration =true;
            errors++;
            System.out.println(ctx.ID().getText()+" ya esta declarada");
        }else{
            tempMemory.put(id,null);
        }
    }else if(!memory.containsKey(id)){
        memory.put(id,null);
    }else{
        errorDeclaration =true;
        errors++;
        System.out.println(ctx.ID().getText()+" ya esta declarada");
    }
    return null;
}
```

```java
@Override
public Object visitAsignacion(OpmezParser.AsignacionContext ctx) {
    String id = ctx.ID().getText();

    if(errorDeclaration){
        errors++;
        System.out.println(ctx.ID().getText()+" no esta declarada");
        return null;
    }else{
        try{
            Object value = visit(ctx.expr());
            if(joinIfElse || bodyInScope){
                if(!memory.containsKey(id)){
                    tempMemory.put(id, value);
                    return tempMemory.get(id);
                }else{
                    memory.put(id,value);
                    return memory.get(id);
                }
            }else if(memory.containsKey(id)) {

                memory.put(id, value);
                return memory.get(id);
            }else{
                errors++;
                System.out.println(ctx.ID().getText()+" no esta declarada");
                return null;
            }
        }catch (Exception e){
            System.out.println("Ocurrio un error en la asignacion de: "+id);
            return null;
        }
    }
}
```

El control del scoppe de variables está en la declaración y asignación de estas. En la declaración se evalúa que la variable no exista previamente en la memoria general, si la declaración esta dentro de if por ejemplo, se verifica que no este en la memoria temporal y general, si alguna de estas condiciones no se cumplen ocurre un error en la declaración lo que nos manda a un error del programa.

Conclusión

La implementación de este scope de variables fue con 2 hashmap’s, el uso de cada una beneficia al funcionamiento del programa, una contendrá los valores persistentes que se pueden modificar en cualquier punto del código, mientras que la memoria temporal solo nos ayudara a usar variables dentro de bloques de código especifico y la memoria se libera en cuanto la lectura salga de este bloque.