**Proyecto de Autómatas y Lenguajes**
**Curso 18/19**
**Grupo Miércoles 16:00-18:00**

**Objetivo de la sesión**
Esta sesión tiene los siguientes objetivos

- **Conjuntos previos**
  - Presentar el material disponible
    - Especificación de ómicron
    - Enunciado de TS
    - Más documentación de ómicron
  - Comentario del material por venir
    - LIstado de todos los casos de prueba / corrección
    - .o de nuestra TS para chequear.
- **Procedural**
  - Avance en el desarrollo del main de la práctica de TS
  - Avance en funciones de TS
- **OO**
  - Test de sintáctico
  - Avance en la funcionalidad de la TS

Plan de trabajo

<table>
<tr><td>OO1:<br>Test sintáctico</td><td rowspan="2">Todos de forma conjunta<br>Funciones de búsqueda<br>• para usar símbolos cualificados por instancia y clase<br>• previa a la declaración de miembros de clase y de instancia</td></tr>
<tr><td>PROC + mixto<br>Trabajo en el main de prueba_TS</td></tr>
</table>

**Actividad OO1 : test analizador sintáctico (creo que lo voy a hacer al final) (30 min)**

Sigue las indicaciones de tu profesor para realizar esta prueba

**Actividad P1: programa principal de la práctica de TS**

Por grupos de procedural + mixto avanza en la codificación del programa principal
A continuación tienes unas sugerencia.
En la medida de lo posible se comprobará el progreso de cada uno a lo largo de la clae.

1. Consulta el enunciado
   - En él se reúne la descripción de las funciones de alto nivel de la TS que llevamos discutiendo varias semanas
   - También se te explica que debes escribir un programa principal que
     - lea peticiones a tu tabla de símbolos desde un fichero de entrada
     - llame a las funciones adecuadas
     - genere un mensaje en un fichero de salida
   - Incide en la lectura de la descripción de este programa que es el objetivo de esta actividad

2. Prepara tu programa principal prueba_TS.c

3. Incluye los includes que necesites

4. Define símbolos para cada elemento del fichero de entrada

```
// OPERACIONES

#define TOK_OP_INICIA_TSA_MAIN          "inicia_tsa_main"
#define TOK_OP_ABRIR_AMBITO_PPAL_MAIN   "abrir_ambito_ppal_main"
#define TOK_OP_BUSCAR                   "buscar"
#define TOK_OP_INSERTAR_TSA_MAIN "insertar_tsa_main"
#define TOK_OP_ABRIR_AMBITO_TSA_MAIN    "abrir_ambito_tsa_main"
#define TOK_OP_CERRAR_AMBITO_TSA_MAIN   "cerrar_ambito_tsa_main"
#define TOK_OP_INICIA_TSC        "inicia_tsc"
....
#define TOK_OP_CERRAR_CLASE             "cerrar_clase"
#define TOK_OP_CERRAR_TSA_MAIN          "cerrar_tsa_main"
#define TOK_OP_CERRAR_TSC        "cerrar_tsc"

// PARAMETROS
#define TOK_DECLARAR_MAIN        "declarar_main"
#define TOK_DECLARAR_MIEMBRO_CLASE      "declarar_miembro_clase"
#define TOK_DECLARAR_MIEMBRO_INSTANCIA  "declarar_miembro_instancia"
...
#define TOK_ID_CUALIFICADO_INSTANCIA    "id_cualificado_instancia"
#define TOK_ID_CUALIFICADO_CLASE "id_cualificado_clase"
```

5. Define (si quieres aquí o en el main) las variables para
   - La tablaAmbitos de main
   - La tabla de símbolos de clases para la jeraquía.
   - El puntero al elemento de la tabla de símbolos para recoger el resultado de búsquedas
   - Idem el nombre de ámbito
   - Los ficheros de entrada y salida

6. Prapara tu programa principal para leer el fichero de entrada. Tal vez puedas pensar en la combinación fgets strtok.. .algo así:

```
while( fgets(linea,100,fd_in) != NULL)
{
       token=strtok(linea," \n\t");
}
```

- De hecho un bucle así debería mostrarte la primera palabra de cada “orden” (línea)
- Recuerda que sucesivas llamadas a strtok como

```
token=strtok(NULL," \n\t");
ambito =(char*)malloc(sizeof(char)*(1+strlen(token)));
strcpy(ambito,token);
categoria= atoi(strtok(NULL," \n\t"));
```

- Te permitirían obtener todos los elementos de cada operación (en este caso, el nombre del ámbito y el entero que indica la categoría del elemento insertado)

7. Tal vez te parezca adecuado que tu programa principal llame a una función para cada diferente instrucción tipo

```
void gestiona_INICIA_TSA_MAIN()
void gestiona_ABRIR_AMBITO_PPAL_MAIN()
void gestiona_BUSCAR()
....
void gestiona_INSERTAR_TSA_MAIN ()
void gestiona_ABRIR_AMBITO_TSA_MAIN()
void gestiona_CERRAR_AMBITO_TSA_MAIN()
void gestiona_INICIA_TSC ()
void gestiona_ABRIR_CLASE()
void gestiona_ABRIR_CLASE_HEREDA ()
void gestiona_INSERTAR_TSC()
void gestiona_ABRIR_AMBITO_TSC()
void gestiona_CERRAR_AMBITO_TSC  ()
void gestiona_CERRAR_CLASE ()
void gestiona_CERRAR_TSA_MAIN    ()
void gestiona_CERRAR_TSC  ()
```

8. Si te gusta esta sugerencia, lo que te quedaría es simplemente llamar en el bucle del main, dependiendo de la primera palabra de cada instrucción a la función adecuada

**Actividad OO y PROC 2**

Actividad conjunta de avance en la codificación de la TS

- OOs aprovechad para poner en común con vuestros compañeros Procs lo que neceisten

Dividíos en 2 subgrupos en los que haya un OO
El trabajo va a consistir en lo siguiente

- Nos centraremos en las parejas de funciones
  - Gruop 1: `buscarParaDeclararMiembroClase` y `buscarParaDeclararMiembroInstancia`.
  - Grupo 2: `buscarIdIDCualificadoClase` y `buscarIdCualificadoInstancia`.
- Debes consultar simultaneamente
  - La descripción de las funciones
  - Los siguientes ejemplos ol con comentarios a la mayoría de los casos que te encontrarás (falta alfuno del grupo 2)
- El objetivo en ambos casos es que no te queden dudas de lo que hay que hacer
- Durante la semana deberás ponerlo en común con tus compañeros
- Te recomiendo que uses el programa principal prueba_TS que están desarollando tus compañeros proc para probar estas funciones desde ahora en adelante con ficheros de entrada como los descritos en el enunciado.

*PARA EL GRUPO 1*

Buscar para declarar miembro de clase

```
main {
	// Buscar para Declarar Miembro (atributo o metodo) de una clase
	// Caso 49: Existe en la clase --> ERR
	// Caso 50: No existe --> Se inserta --> OK

	class AA {
		// Declaraciones de atributos
		secret  unique int sA1;			// Caso 50 Buscar: ERROR (AA_sA1) NO
ESTÁ EN AA (SOLO SE BUSCA EN AA) SE PUEDE DECLARAR
		secret unique int sA1;			// Caso 49 BUSCAR: OK (AA_sA1) YA ESTÁ
EN AA (SOLO SE BUSCA EN AA) NO SE PUEDE DECLARAR
		hidden unique int hA1;			// Caso 50 BUSCAR: ERROR (AA_hA1) NO
ESTÁ EN AA (SOLO SE BUSCA EN AA) SE PUEDE DECLARAR
		exposedunique int xA1;			// Caso 50 BUSCAR: ERROR (AA_xA1) NO
ESTÁ EN AA (SOLO SE BUSCA EN AA) SE PUEDE DECLARAR

		// Metodos
		function hidden  unique int MA1 (int p1) { int v1; printf p1;}
// Caso 49 BUSCAR: ERROR (AA_mA1@1) NO ESTÁ EN AA (SOLO SE BUSCA EN AA) SE PUEDE
DECLARAR
		function secret  unique int MA1 (int p1) { int v1; printf p1+v1;} // Caso
50 BUSCAR: OK (AA_mA1@1) ESTÁ EN AA (SOLO SE BUSCA EN AA) NO SE PUEDE DECLARAR
		function hidden  unique int MA1 (int p1;int p2) { int v1; printf p1;}
// Caso 49 BUSCAR: ERROR (AA_mA1@1@1) NO ESTÁ EN AA (SOLO SE BUSCA EN AA) SE PUEDE
DECLARAR COMO UN NUEVO MÉTODO DE CLASE QUE REALMENTE ESTÁ SOBRECARGANDO OTRO (AA_mA1@1)
	};

	class BB {
		// Declaraciones de atributos
		secret  unique int sA1;			// Caso 50 Buscar: ERROR (BB_sA1) NO
```

```
ESTÁ EN BB (SOLO SE BUSCA EN BB) SE PUEDE DECLARAR
              hidden unique int hA1;              // Caso 50 Buscar: ERROR (BB_hA1) NO
ESTÁ EN BB (SOLO SE BUSCA EN BB) SE PUEDE DECLARAR
              exposedunique int xA1;              // Caso 50 Buscar: ERROR (BB_xA1) NO
ESTÁ EN BB (SOLO SE BUSCA EN BB) SE PUEDE DECLARAR

              // Metodos
              function secret  unique int MB1 (int p1) { int v1; printf p1+v1;} // Caso
50 Buscar: ERROR (BB_mB1) NO ESTÁ EN BB (SOLO SE BUSCA EN BB) SE PUEDE DECLARAR
              function hidden  unique int MA1 (int p1) { int v1; printf p1;}
// Caso 50 Buscar: ERROR (BB_mA1) NO ESTÁ EN BB (SOLO SE BUSCA EN BB) SE PUEDE DECLARAR
              function exposed unique int MA2 (int p1;int p2) { int v1; printf p1;}
// Caso 50 Buscar: ERROR (BB_mA2) NO ESTÁ EN BB (SOLO SE BUSCA EN BB) SE PUEDE DECLARAR
       };

       {AA} miA;
       {BB} miB;

       miA = instance_of AA();

       discard miA;

}
```

## Buscar para declarar miembro instancia

```
main {
       // Buscar para Declarar Miembro (atributo o metodo) de una clase
       // Caso 51: Existe en la clase --> ERR
       // Caso 52: No existe en la clase, pero si en la jerarquia --> depende de los
accesos
       // Caso 53: No existe --> Se inserta --> OK

       class AA {
              // Declaraciones de atributos
              secret  int sa1;             // Caso 53 Buscar: ERROR (AA_sa1) SE PUEDE
DECLARAR
              hidden  int sa1;             // Caso 51 Buscar: OK (AA_sa1) NO SE PUEDE
DECLARAR
              secret  int ha1;             // Caso 53 Buscar: ERROR (AA_ha1) SE PUEDE
DECLARAR
              exposed boolean xa1;  // Caso 53 Buscar: ERROR (AA_xa1) SE PUEDE DECLARAR

              // Metodos
              function secret  int mA1 (int p1) { int v1; printf p1;}   // Caso 53
Buscar: ERROR (AA_mA1@1) SE PUEDE DECLARAR
                                                                        // COMO NUEVO
MÉTODO SOBREESCRIBIBLE
              function hidden  int mA1 (int p1) { int v1; printf p1+v1;}// Caso 51
Buscar: OK (AA_mA1@1) NO SE PUEDE DECLARAR
              function secret  int mA1 (int p1; boolean p2) { printf p2;} // Caso 53
Buscar: ERROR (AA_mA1@1@3) NO EXISTE, EXISTE AA_mA1@1
                                                                  // SE PUEDE
DECLARAR COMO NUEVO MÉTODO SOBREESCRIBIBLE: EN REALIDAD
                                                                  // SE ESTÁ
SOBRECARGANDO
              function hidden  int mA2 () { int v1; printf v1;}  // Caso 53 Buscar:
ERROR (AA_mA2) SE PUEDE DECLARAR
              function exposed int mA3 (int p1; boolean p2) { int v1; int v2; printf
v1+v2;}       // Caso 51 Buscar: ERROR (AA_mA3@1@3)

// SE PUEDE DECLARAR

       };

       class BB inherits AA { // En todos estos casos en los que no está definido en el
```

```
ámbito actual (clase) pero sí en el padre
                            // todo depende de los accesos y del tipo de atributo que se
quiera instalar, hay que mirar los retornos
                            // de la función de búsqueda
              secret  int sa1;      // Caso 52: Buscar: OK (AA_sa1) DECLARADO EN AA ES
ATRIBUTO DE INSTANCIA ACCESIBLE CON ACCESO secret NO
                                    // SE PUEDE DECLARAR
              hidden  int ha1;      // Caso 52: Buscar: ERROR (BB_ha1) EXISTE AA_ha1 (EN
AA) PERO NO ES ACCESIBLE PORQUE ES HIDDEN
                                    // SE PUEDE DECLARAR
              hidden  int hb1;      // Caso 53: Buscar: ERROR (BB_hb1) NO EXSITE, SE
PUEDE DECLARAR
              exposed int xa1;      // Caso 52: Buscar: OK (AA_xa1) DECLARADO EN AA ES
ATRIBUTO DE INSTANCIA ACESIBLE CON ACCESO exposed NO
                                    // SE PUEDE DECLARAR

              // Metodos REVISAR ESTOS
              function secret  int mA1 (int p1) { int v1; printf p1;}    // Caso 52:
Buscar: OK (AA_mA1@1) EXISTE EN LA JERARQUIA (AA)
                                                                         // ES ACCESIBLE
(SECRET) Y ES SOBREESCRIBIBLE: SE ESTÁ

//SOBREESCRIBIENDO Y EL OFFSET ACUMULADO PAR LA TABLA DE
                                                                         // MÉTODOS ES EL
MISMO QUE EL DE AA_mA1@1
              function secret  int mB1 () { int v1; v1 = 8; printf v1;} // Caso 53:
BUSCAR: ERROR (BB_mB1) NO EXISTE EN NINGÚN LUGAR
                                                                         // SE PUEDE
DECLARAR
              function hidden  int mA2 () { boolean v1; printf v1;}             // Caso
52: BUSCAR: ERROR (AA_mA2) mA2 EXISTE EN AA PERO NO
                                                                         // ES ACCESIBLE
(HIDDEN) SE PEUDE DECLARAR COMO UN MÉTODO
                                                                         //
SOBRERESCRBIBLE NUVO CON SU PROPIO OFFSET ACUMULADO PARA
                                                                         // LA TABLA DE
METODOS
              function exposed int mA3 (boolean p1; int p2) { printf p2;}       // Caso
53: BUSCAR: ERROR (BB_mB1) NO EXISTE EN NINGÚN LUGAR
                                                                         // SE PUEDE
DECLARAR, DE HECHO SE ESTA SOBRECARGANDO AA_mA3@1@3
                                                                         // QUE TENÍA LOS
ARGUMENTOS EN DISTINTO ORDEN
       };

       class BB inherits AA {                       // ERR
       };

       {AA} miA;
       {BB} miB;

       miA = instance_of AA();
       miB = instance_of BB();

       discard miA;
       discard miB;
}
```

*PARA EL GRUPO 2*

Buscar para uso de identificador cualificado clase / instancia

```
main {

int varGlob;



//Declaraciones.

class AA {
    exposed {AA} inspub;
//    hidden {AA} insprot;
//    secret {AA} inspriv;
    secret {AA} insprot;
    hidden {AA} inspriv;


    exposed unique {AA} clasepub;
//    hidden unique {AA} claseprot;
//    secret unique {AA} clasepriv;
    secret unique {AA} claseprot;
    hidden unique {AA} clasepriv;


    exposed {AA} miaa;

    function secret int prueba ()
    {
        //CASO 47.3 SE ACCEDE DESDE UN MÉTODO A UN ATRIBUTO QUE EXISTE EN LA JERARQUÍA Y
ESTÁ ACCESIBLE (inspub es exposed e insprot es secret)
        // CUALIFICANDO POR UNA INSTANCIA ACCESIBLE (miaa) ==> DEBE DEVOLVER OK
        //CASO 48. SE ACCEDE DESDE UN MÉTODO A UN ATRIBUTO (inspriv) QUE EXISTE EN LA
JERARQUÍA (AA) Y NO ESTÁ ACCESIBLE (inspriv es hidden)
        // A TRAVÉS DE UNA INSTANCIA QUE SÍ ESTÁ ACCESIBLE (miaa) ==> DEBE DEVOLVER ERR
         printf miaa.inspriv + miaa.insprot+ miaa.inspub;

        //CASO 35. SE BUSCA DESDE UN MÉTODO ALGO DE UNA CLASE (AA) QUE ESTÁ EN LA
JERARQUÍA Y ES ACCESIBLE (claseprot es secret
         // y estamos en la clase y clasepub es exposed) ==> DEBE DEVOLVER OK
         if ( (AA.claseprot == AA.clasepub))
         {
             AA.clasepriv = AA.clasepub;
         }
         return 0;
    }
};

class BB inherits AA {
    exposed {BB} mibb;

};


//Declaracion de variables globales.
{BB} MainMiBB;
{BB} MainMiBB2;

//CASO 40.1. SE BUSCA ALGO QUE NO EXISTE CUALIFICADO POR UNA INSTANCIA QUE EXISTE Y ES
GLOBAL ==> DEBE DEVOLVER ERR
printf MainMiBB.insnoexiste;

class CC {
    exposed {CC} micc;
};

class DD inherits AA, CC {
    exposed {DD} midd;
};

class EE inherits BB, DD {
    exposed {EE} miee;
```

```
    function exposed none mE(){
       {EE} varLoc;

       //CASO 36. DESDE DENTRO DE UN MÉTODO, SE ACCEDE A ALGO CUALIFICADO POR UNA CLASE
(E) QUE EXISTE PERO LO
       // BUSCADO ESTÁ EN LA JERARQUIA Y NO ES ACCESIBLE (clasepriv)  ==> DEBE DEVOLVER
ERR
        if ( (EE.clasepub == EE.clasepriv))
        {
       //CASO 37. DESDE DENTRO DE UN MÉTODO, SE ACCEDE A ALGO CUALIFICADO POR UNA CLASE
(E) QUE EXISTE PERO LO
       // BUSCADO NO EXISTE (claseNoExiste)  ==> DEBE DEVOLVER ERR
            EE.claseNoExiste = EE.clasepub;
        }
        //CASO 38. DESDE DENTRO DE UN MÉTODO, SE ACCEDE A ALGO CUALIFIACADO POR UNA
CLASE (noExiste) QUE NO EXISTE
       // ==> DEBE DEVOLVER ERR
        printf     noExiste.claseNoExiste;

       //CASO 46.3: DESDE DENTRO DE UN MÉTODO, SE BUSCA ALGO CUALIFICADO POR UNA
INSTANCIA DE UNA CLASE (B) QUE ES UNA VARIABLE GLOBAL
       // Y LO BUSCADO ES UN ATRIBUTO (inspub) QUE NO ESTÁ EN LA CLASE PERO SÍ EN LA
JERARQUÍA (en A) Y ES ACCESIBLE (exposed)
       // ==> DEBERÍA DEVOLVER OK
        printf MainMiBB2.inspub;

       //CASO 47.1: DESDE UN MÉTODO SE ACCEDE A ALGO CUALIFICADO POR UNA INSTANCIA
(miee) QUE ES UN ATRIBUTO ACCESIBLE EN ESTE CASO DE LA
       // MISMA CLASE PERO EL ATRIBUTO BUSCADO (insNoExiste) NO EXISTE ==> DEBE DEVOLVER
ERR
        printf miee.insNoExiste;

       //CASO 45.3: DESDE UN METODO SE ACCEDE A ALGO CUALIFICADO POR UNA INSTANCIA
(varLoc) QUE ES UNA VARIABLE LOCAL DEL MÉTODO
       // DE UN CLASE QUE EXISTE (E) Y LA COSA BUSCADA (inspub) ES UNA ATRIBUTO QUE
AUNQUE NO ESTÁ EN LA CLASE SÍ ESTÁ EN
       // LA JERARQUÍA (A) Y ES ACCESIBLE (exposed) ==> DEBE DEVOLVER OK Deberia
funcionar.
        printf varLoc.inspub;

        printf (miee.inspub==miee.insprot);

        return none;
    }
};

class FF inherits CC, EE {
    exposed {FF} miff;
};

//Algunas pruebas desde el main.

function int prueba() {

    //CASOS 32 Y 31: ACCESOS DESDE UNA FUNCIÓN GLOBAL (prueba) A ALGO CUALIFICADO POR
UNA CLASE (A) QUE EXISTE Y LO BUSCADO
    // RESEPCTIVAMENTE (clasepriv Y clasepub) SON ATRIBUTOS DE ESA CLASE SON DE CLASE Y
EL PRIMERO NO ES ACCESIBLE Y EL
    // SEGUNDO SI ==> DEBE DEVOLVER 32 OK Y 31 ERR
    if ( (AA.clasepriv == AA.clasepub))
    {
       //CASO 33: ACCESO DESDE UNA FUNCIÓN GLOBAL (prueba) A ALGO CUALIFICADO POR UNA
CLASE (A) QUE EXISTE Y LO
       // BUSCADO (claseNoExiste) NO ==> DEBE DEVOLVER ERR
        AA.clasepriv = AA.claseNoExiste;
    }

    //CASO 34: ACCESO DESDE FUNCIÓN GLOBAL DE ALGO CUALIFICADO POR UNA CLASE QUE NO
EXISTE ==> DEBE DEVOLVER ERROR
```

```
    printf noExiste.claseNoExiste;


    //CASO 43.1. ACCESO DESDE UNA FUNCIÓN GLOBAL (prueba) DE UN ID (insNoExiste)
CUALIFICADO POR UNA INSTANCIA (MainMiBB2)
    // QUE EXISTE COMO VARIABLE GLOBAL DE UNA CLASE (B) QUE NO TIENE ESE ATRIBUTO DE
NINGUNA MANERA ==> DEBE DEVOLVER ERR
    if ((MainMiBB2.inspriv == MainMiBB2.insNoExiste))
    {
        printf 0;
    }
    return 0;
}

//Pruebas con una funcion global.
function secret none f2 (){

    int varLoc;
    {BB} MainMiBB3;

    //CASO 43.1: ACCESO DESDE UNA FUNCIÓN GLOBAL (f2) DE UN ID (insNoExiste) CUALIFICADO
POR UNA INSTANCIA (MainMiBB2)
    // QUE EXISTE COMO VARIABLE GLOBAL DE UNA CLASE (B) QUE NO TIENE ESE ATRIBUTO DE
NINGUNA MANERA ==> DEBE DEVOLVER ERR
    printf MainMiBB2.insNoExiste;

    //CASO 43.3: ACCESO DESDE UNA FUNCIÓN GLOBAL (f2) DE UN ID (inspub y inspriv)
CUALIFICADO POR UNA INSTANCIA (MainMiBB2)
    // QUE EXISTE COMO VARIABLE GLOBAL DE UNA CLASE (B) QUE NO TIENE ESOS ATRIBUTOS PERO
QUE SÍ LOS TIENE SU JERARQUIA
    // (A) Y ADEMÁS DESDE ELLA (B) SON ACCESIBLES (RESPECTIVAMENTE SON EXPOSED Y SECRET
Y B HEREDA DE A) ==> DEBE DEVOLVER OK
    printf MainMiBB2.inspub;
    if ( (MainMiBB2.inspub == MainMiBB2.inspriv) )
    {
        printf 0;
    }

    //CASO 31.1 YA DESCRITO ==> DEBE DEVOLVER OK
    printf AA.clasepub;

    //CASO 32. YA DESCRITO ==> DEBE DEVOLVER ERR
    printf AA.clasepriv;

    //CASO 42.3 DESDE UNA FUNCIÓN GLOBAL (f2) SE ACCEDE A UN ID CUALIFICADO POR UNA
INSTANCIA QUE ES UNA VARIABLE GLOBAL (MainMiBB2)
    // DECLARAD DE TIPO DE CLASE QUE EXISTE (B) Y EL ID (inspub) ES UN ATRIBUTO QUE
EXISTE EN LA JERARQUIA (A) Y ES ACCESIBLE ==> DEBE
    // DEVOLVER OK
    printf MainMiBB2.inspub;
    return 0;
}


    //PRUEBAS DESDE EL MAIN.

    varGlob = 0;

    //CASOS 30: DESDE EL CÓDIGO DE MAIN SE BUSCA ALGO CUALIFICADO POR UNA CLASE QUE NO
EXISTE (JJ) ==> DEBE DEVOLVER ERR
    printf JJ.noExiste;

    //CASO 40.3: DESDE EL CÓDIGO DE MAIN SE BUSCA ALGO CUALIFICADO POR UNA INSTANCIA QUE
ES UNA VARIABLE GLOBAL (MainMiBB2)
    // DECLARAD DE TIPO DE CLASE QUE EXISTE (B) Y EL ID (inspub) ES UN ATRIBUTO QUE
EXISTE EN LA JERARQUIA (A) Y ES ACCESIBLE ==> DEBE
    // DEVOLVER OK
    printf MainMiBB2.inspub;

    //CASO 27.1: DESDE EL CÓDIGO DE MAIN SE BUSCA ALGO CUALIFICADO POR UNA CLASE QUE
```

```
EXISTE (AA) Y EL ID BUSCADO (clasepub) ES
    // UN ATRIBUTO QUE EXISTE EN LA CLASE, ACCESIBLE Y ATRIBUTO DE CLASE ==> DEBE
DEVOLVER OK
    printf AA.clasepub;

    //CASO 28: DESDE EL CÓDIGO DE MAIN SE BUSCA ALGO CUALIFICADO POR UNA CLASE QUE
EXISTE (AA) Y EL ID BUSCADO (clasepub) ES
    // UN ATRIBUTO QUE EXISTE EN LA CLASE, NO ACCESIBLE (HIDDEN) ==> DEBE DEVOLVER ERR
    printf AA.clasepriv;

    //CASO 29: DESDE EL CÓDIGO DE MAIN SE BUSCA ALGO CUALIFICADO POR UNA CLASE QUE
EXISTE (AA) Y EL ID BUSCADO (noExiste) NO EXISTE
    // ==> DEBE DEVOLVER ERR
    printf AA.noexiste;

    //CASO 30 ==> YA PROBADO DEBE DEVOLVER ERR
    printf Noexiste.noexiste;
}
```

**PARA LA PRÓXIMA SEMANA**

**PP+OO+mixto**

- Incorporar todo lo visto
- Poner al día a los PP de la parte OO hasta el momento