2) Compilación. El lenguaje CALCPROG2 (2 puntos)

Supongamos un lenguaje de programación llamado CALCPROG2 para evaluar expresiones enteras. El programa CALCPROG2 consta de una secuencia de órdenes y un conjunto de funciones. Hay 3 tipos de órdenes en CALCPROG2: expresión, impresión por pantalla de valores de expresiones y asignación.

Para precisar y completar la definición sintáctica del lenguaje, se adjunta su gramática a continuación:

```
programa: ordenes funciones

ordenes: ORDENES DOSPUNTOS decl_ordenes

decl_ordenes: orden decl_ordenes
   | orden

funciones: FUNCIONES DOSPUNTOS decl_funciones

decl_funciones: funcion decl_funciones
   | funcion

orden: expresion PUNTOYCOMA | asignacion | impresion

funcion: cabecera ASIGNACION expresion PUNTOYCOMA

cabecera: IDENT PARENTESISABIERTO variables PARENTESISCERRADO

variables: IDENT COMA variables
       | IDENT

asignacion: IDENT ASIGNACION expresion PUNTOYCOMA

impresion: MOSTRAR PARENTESISABIERTO expresiones PARENTESISCERRADO PUNTOYCOMA

expresiones: expresion COMA expresiones
       | expresion

expresion: expresion MAS expresion
   | expresion MENOS expresion
   | expresion POR expresion
   | expresion DIVISION expresion
   | PARENTESISABIERTO expresion PARENTESISCERRADO
   | NUMERO
   | IDENT PARENTESISABIERTO expresiones PARENTESISCERRADO
   | IDENT
```

A continuación, se muestra un programa de ejemplo CALCPROG2:

```
ORDENES:
    mostrar(3 + (4 + 1));
    a = 1 + 1;
    b = a;
    mostrar(a,f(a));
    mostrar(g(3,b));
    f(f(2));
    mostrar(f(a+1));

FUNCIONES:
   f(a) = 10 * a;
   g(x,y) = 10*f(x) + y;
   i(x) = x;
```

SE PIDE:

Decisiones y gramática atribuida para compilador de CALCPROG2 a Java.

```
OBJETIVO
--------
   Construir un compilador para el lenguaje Calcprog2.


DECISIONES
----------
   (1) El programa Calcprog2 se escribe en un fichero con extensión java
       Este fichero declara una clase no instanciable con un main para la
       secuencia de órdenes y métodos éstaticos para las funciones.

       Ejemplo:      ORDENES:    ...   FUNCIONES: f(...)  ...

               se traduce como:

                     public class Nombre_Clase{
                        public static void main(String[] args){
                          ...
                        }
                        static Integer f(...)
                        ...
                     }
   (2) Dado que Calcprog2 no declara explícitamente variables, se necesita una memoria
       (llamada variables_declaradas) para almacenar las variables que ya se han
       declarado en el main().

   (3) Generar código Java de forma recursiva para las expresiones:

   expresion dev cod:
      cod1=expresion MAS cod2=expresion {cod=cod1+"+"+cod2}
      | cod1=expresion MENOS cod2=expresion {cod=cod1+"-"+cod2}
      | cod1=expresion POR cod2=expresion {cod=cod1+"*"+cod2}
      | cod1=expresion DIVISION cod2=expresion {cod=cod1+"/"+cod2}
      | PARENTESISABIERTO cod1=expresion PARENTESISCERRADO
        {cod="("+cod1+")"}
      | NUMERO {cod = NUMERO}
      | IDENT PARENTESISABIERTO cod1=expresiones PARENTESISCERRADO
        {cod=IDENT+"("+cod1+")"}
      | IDENT {cod=IDENT}
```

```
   (4) Generar código Java para las asignaciones.
      asignacion: IDENT ASIGNACION cod=expresion PUNTOYCOMA
      { si IDENT ha sido previamente decladara entonces
           escribir en fichero: IDENT+" = "+cod+";"
        sino
           escribir en fichero: "Integer "+IDENT+" = "+cod+";"
        fsi }

   (5) Generar código para la órden de impresión:
       impresion: MOSTRAR PARENTESISABIERTO cod=expresiones PARENTESISCERRADO
PUNTOYCOMA
       { escribir en fichero: "System.out.println("+cod+")" }

   (6) Generar métodos estáticos para las funciones:

       funcion: cod1=cabecera ASIGNACION cod2=expresion PUNTOYCOMA
       { escribir en fichero: cod1+"{"
         escribir en fichero:    "return "+cod2+";"
         escribir en fichero: "}"
       }

       cabecera dev cod: IDENT PARENTESISABIERTO cod1=variables PARENTESISCERRADO
          {  cod = IDENT+"("+cod1+")" }


GRAMÁTICA ATRIBUIDA
-------------------
(global)
    variables_declaradas
    fichero


    abrir_fichero(nombre_clase)
    cerrar_fichero(fichero)

    generar_codigo_cabecera_clase(nombre_clase){
       escribir en fichero el texto:
            "import java.util.*;
             public class "+nombre_clase+"{"
    }

    generar_codigo_cabecera_main(){
       escribir en fichero el texto: "   public static void main(String[] args){"
    }

    generar_codigo_fin_clase(){
       escribir en fichero el texto: "}"
    }

    generar_codigo_fin_funcion(){
       escribir en fichero el texto: "   }"
    }
```

```
////////////////////// REGLAS ////////////////////////////

programa[nombre_clase]:
    { fichero = abrir_fichero(nombre_clase)
      generar_codigo_cabecera_clase(nombre_clase) }
      generar_codigo_cabecera_main() }
             ordenes
      { generar_codigo_fin_funcion() }
             funciones
     { generar_codigo_fin_clase()
       cerrar_fichero(fichero) }

ordenes: ORDENES DOSPUNTOS decl_ordenes

decl_ordenes: orden decl_ordenes
   | orden

funciones: FUNCIONES DOSPUNTOS decl_funciones

decl_funciones: funcion decl_funciones
   | funcion

orden: cod=expresion PUNTOYCOMA { escribir en fichero: "      "+cod+";"}
     | asignacion
     | impresion

funcion: cod1=cabecera ASIGNACION cod2=expresion PUNTOYCOMA
   { escribir en fichero: "   "+cod1+"{"
     escribir en fichero: "      return "+cod2+";"
     escribir en fichero: "   }"
   }

cabecera dev cod: IDENT PARENTESISABIERTO cod1=variables PARENTESISCERRADO
   {  cod = "public static Integer "+IDENT+"("+cod1")" }

variables dev cod:
       IDENT COMA cod2=variables { cod = "Integer "+IDENT+","+cod2 }
     | IDENT { cod = "Integer "+IDENT }

asignacion: IDENT ASIGNACION cod=expresion PUNTOYCOMA
  { si IDENT ha sido previamente decladara entonces
       escribir en fichero: "      "+IDENT+" = "+cod+";"
    sino
       almacenar IDENT en variables_declaradas
       escribir en fichero: "      Integer "+IDENT+" = "+cod+";"
    fsi }

impresion: MOSTRAR PARENTESISABIERTO cod=expresiones[impresion] PARENTESISCERRADO
PUNTOYCOMA
       { escribir en fichero: "System.out.println("+cod+");" }

expresiones[contexto] dev cod:
    cod1=expresion COMA cod2=expresiones
    { si contexto es impresion entonces
          cod = cod1+"+\",\"+"+cod2
      sino
          cod = cod1+","+cod2
      }
  | cod=expresion
```

```
expresion dev cod:
      cod1=expresion MAS cod2=expresion {cod=cod1+"+"+cod2}
      | cod1=expresion MENOS cod2=expresion {cod=cod1+"-"+cod2}
      | cod1=expresion POR cod2=expresion {cod=cod1+"*"+cod2}
      | cod1=expresion DIVISION cod2=expresion {cod=cod1+"/"+cod2}
      | PARENTESISABIERTO cod1=expresion PARENTESISCERRADO
        {cod="("+cod1+")"}
      | NUMERO {cod = NUMERO}
      | IDENT PARENTESISABIERTO cod1=expresiones[expresion] PARENTESISCERRADO
        {cod=IDENT+"("+cod1+")"}
      | IDENT {cod=IDENT}
```

Muestre la traducción que genera su solución para el programa de ejemplo.

```
import java.util.*;
public class Compilador1{
   public static void main(String[] args){
      System.out.println(3+(4+1));
      Integer a = 1+1;
      Integer b = a;
      System.out.println(a+","+f(a));
      System.out.println(g(3,b));
      f(f(2));
      System.out.println(f(a+1));
   }
   public static Integer f(Integer a){
      return 10*a;
   }
   public static Integer g(Integer x,Integer y){
      return 10*f(x)+y;
   }
   public static Integer i(Integer x){
      return x;
   }
}
```

NOTA: sea claro y legible.