2) Compilación. El lenguaje L (2 puntos)

L es un lenguaje de programación secuencial con tipado dinámico de sus variables (las variables toman el tipo de las expresiones asignadas). El programa L está compuesto por una secuencia de instrucciones (asignación y muestra de valores de variables por consola). Las expresiones en L son de dos tipos: (a) enteras y (b) listas. Los elementos de una lista pueden ser enteros u otras listas. Las expresiones enteras, además de las operaciones convencionales, incluyen la selección de elementos enteros de una lista. Las expresiones listas incluyen listas por extensión, segmentos de listas definidos con rangos de índices y selección de elementos listas de una lista. Para precisar y completar la definición sintáctica del lenguaje, se adjunta su gramática a continuación:

```
programa: PROGRAMA instrucciones FPROGRAMA

instrucciones: instruccion instrucciones
   | instruccion

instruccion: asignacion | mostrar

asignacion: IDENT ASIGNACION expresion PUNTOYCOMA

mostrar: MOSTRAR PARENTESISABIERTO variables PARENTESISCERRADO PUNTOYCOMA;

variables: variable COMA variables
     | variable

expresiones: expresion COMA expresiones
       | expresion

expresion: expresion_entera
   | expresion_lista


expresion_entera: expresion_entera MAS expresion_entera
   | expresion_entera MENOS expresion_entera
   | expresion_entera POR expresion_entera
   | expresion_entera DIVISION expresion_entera
   | PARENTESISABIERTO expresion_entera PARENTESISCERRADO
   | NUMERO
   | variable

expresion_lista: CORCHETEABIERTO (expresiones)? CORCHETECERRADO
   | PARENTESISABIERTO expresion_lista PARENTESISCERRADO
   | segmento_lista
   | variable

segmento_lista: IDENT CORCHETEABIERTO rango CORCHETECERRADO

rango: expresion_entera RANGO expresion_entera

variable : IDENT CORCHETEABIERTO expresion_entera CORCHETECERRADO
   | IDENT
```

A continuación, se muestra un programa L de ejemplo:

```
PROGRAMA
  x = 1;
  mostrar(x);              // se muestra por consola x = 1
  x = [3,x,12,[1,x+5]];
  y=x[1]+2;
  mostrar(x,y);            //se muestra por consola x = [3,1,12,[1,6]] y = 5
  y = x[x[1]-2..x[1]];
  mostrar(y);              //se muestra por consola y = [3,1]
FPROGRAMA
```

SE PIDE: Completar la gramática atribuida propuesta en la siguiente solución para compilador de L a Java.

NOTA: sea claro y legible.

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


DECISIONES
----------
   (1) El programa L se se traduce como una clase no instanciable con un solo main.
       Ejemplo:

            PROGRAMA    ...   FPROGRAMA   se traduce como:

            import java.util.*;
            public class nombre_de_la_clase_java_dado_al_programa{
               public static void main(String[] args){
                           ...
               }
            }

   (2) Las listas L se traducen a listas Java identificadas con variables frescas
      (variables nuevas en el programa Java).
       Por ejemplo, la lista [3,x] se traduce como:

            List<Object> _aux_1 = new LinkedList<>();
            _aux_1.add(3);
            _aux_1.add(x);
       siendo _aux_1 una variable fresca.

   (3) El tipado dinámico de L se simula con variables Java tipo Object.
       En Java, las variables se declaran antes de usarlas (se declaran una sola vez).
       Ejemplo:
           ...
           x = 1;
           ...
           x = [];

        se traduce como:
```

```
Object x;  //declaración única de x como Object para simular tipado dinámico
...
x = 1;    //x almacena un entero
...
List<Object> _aux_1 = new LinkedList<>();
x=_aux_1; //x almacena una lista vacía
```

(4) Memoria (llamada variables_declaradas) para almacenar las variables que ya se han declarado en el main(). Esta memoria es necesaria para generar código sin errores por redeclaración de variable.

(5) La selección de un elemento de una lista L se traduce con la función Java predefinida get(). Hay que considerar la coerción explícita para poder aplicar dicha función.
Por ejemplo, z[1] se traduce como ((List<Object>)z).get(1-1)

(6) El segmento de lista se traduce con la función Java predefinida sublist (sublist se aplica a List<Object>).
Los índices de las listas L comienzan en 1. Es decir, el elemento 1 en L se corresponde con el 0 en Java.
Por ejemplo, x[1..x[1]] se traduce como:
((List<Object>)x).subList(1-1,(Integer)((List<Object>)x).get(1-1)-1)

(7) Las variables necesitan conversiones explícitas de tipos al usarse en el contexto de expresiones con operadores aritméticos.
Sería necesario el uso de centinela para indicar que una subexpresión necesita conversión explícita de tipo.
Ejemplo: x+1+y se traduce como (Integer)x+1+(Integer)y
sin embargo la traducción de x no necesita conversión explícita.

GRAMÁTICA ATRIBUIDA

```
(global)
    variables_declaradas
    contador_variables_frescas
    fichero

    abrir_fichero(nombre_clase) dev fichero

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

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

programa:
     { fichero = abrir_fichero(nombre_clase)
       generar_codigo_cabecera_clase(nombre_clase) }
     PROGRAMA
        { generar_codigo_cabecera_main()
          inicializar contador_variables_frescas }
        instrucciones
        { generar_codigo_fin_main() }
     FPROGRAMA
     { generar_codigo_fin_clase()
       cerrar_fichero(fichero) }

instrucciones: instruccion instrucciones
   | instruccion

instruccion: asignacion | mostrar

asignacion: IDENT ASIGNACION cod_exp=expresion[no operador] PUNTOYCOMA
   { si IDENT no está en variables declaradas entonces
        almacenar IDENT en variables_declaradas
        escribir en fichero el texto: "      Object "+IDENT+" = "+cod_exp+";"
     sino
        escribir en fichero el texto: "      "+IDENT+" = "+cod_exp+";"
     fsi  }

mostrar: MOSTRAR PARENTESISABIERTO cod=variables PARENTESISCERRADO PUNTOYCOMA
   { escribir en fichero el texto: System.out.println(cod); }

variables dev cod: cod1=variable[no operador] COMA cod2=variables
                   {  cod = cod1+"+\", \""+cod2  }
     | cod=variable[no operador]


expresion[contexto] dev cod: cod=expresion_entera[contexto]
   | cod=expresion_lista

expresion_entera[contexto] dev cod:
     cod1=expresion_entera[operador] MAS cod2=expresion_entera[operador]
     { cod=cod1+"+"+cod2 }
   | cod1=expresion_entera[operador] MENOS cod2=expresion_entera[operador]
     { cod=cod1+"-"+cod2 }
   | cod1=expresion_entera[operador] POR cod2=expresion_entera[operador]
     { cod=cod1+"*"+cod2 }
   | cod1=expresion_entera[operador] DIVISION cod2=expresion_entera[operador]
     { cod=cod1+"/"+cod2 }
   | PARENTESISABIERTO cod1=expresion_entera[contexto] PARENTESISCERRADO
     { cod="("+cod1+")" }
   | NUMERO { cod = NUMERO }
   | cod=variable[contexto]
```

```
expresion_lista dev cod:  { variable_fresca = "_aux_"+contador_variables_frescas
                            incrementar contador_variables_frescas
                            escribir en fichero el texto: "      List<Object>
"+variable_fresca+" = new LinkedList<>();"
                            cod = variable_fresca } CORCHETEABIERTO
(expresiones[variable_fresca])? CORCHETECERRADO
   | PARENTESISABIERTO cod1=expresion_lista PARENTESISCERRADO { cod="("+cod1+")" }
   | cod=segmento_lista
   | cod=variable[no operador]


expresiones[variable_fresca]:
    cod1=expresion[no operador] { escribir en fichero el texto: "
"+variable_fresca+".add("+cod1+");" }
         COMA expresiones[variable_fresca]
  | cod1=expresion[no operador] { escribir en fichero el texto: "
"+variable_fresca+".add("+cod1+");" }

segmento_lista dev cod : IDENT CORCHETEABIERTO cod1,cod2=rango CORCHETECERRADO
   { cod = "((List<Object>)"+IDENT+").subList("+cod1+"-1,"+cod2+"-1)" }


rango dev cod1, cod2:
   cod1=expresion_entera[operador] RANGO cod2=expresion_entera[operador]


variable[contexto] dev cod :
     IDENT CORCHETEABIERTO cod1=expresion_entera[contexto] CORCHETECERRADO
        {  si contexto es operador entonces
              cod="(Integer)((List<Object>)"+IDENT+").get("+cod1+"-1)" }
           sino
              cod="((List<Object>)"+IDENT+").get("+cod1+"-1)"
           fsi }
   | IDENT {si contexto es operador entonces
               cod = "(Integer)"+IDENT
            sino
               cod = IDENT
            fsi}
```