- Symbol Table: it is essentially a wrapper over the HashTable class

  Class arguments: size (int), by default 50 -> fixes the size of the array list of the hashtable

  Class methods:

  addIdentifier(idName: String) -> public method that calls the addItem method from the hashtable object with the identifier name as its argument; it adds the item to hashtable and returns the position which is now occupied by the newly added item

  getIdentifier (idName: String) –> public method that calls the getItem method from the hashtable object with the identifier name as its argument; it returns the item's position in the hashtable if it exists, null otherwise

  toString() - > formats the content of the class in a pretty manner

- HashTable:

  Class arguments: size (int) -> it is passed from symbol table class

  Class methods:

  hashItem (elem T): Int -> it hashes the element into an integer smaller than the size of the hashtable

  getItem (elem T): Pair (Int, Int) -> checks if the elem exists and returns its position in the array as a pair of integers

  addItem (elem T): Pair (Int, Int) -> checks if the elem already exists and if so it returns its position, otherwise it adds the element on the first free position from the sublist found on the hash's result row

- Scanner:
  Class arguments: no arguments

  Class fields: 1 identifiers SymbolTable, 1 constants SymbolTable, 1 PIF
  PIF = an arraylist that consists of pairs of each item that is not a token and its position in the symboltable (identifiers, respectively constants)

  Class methods:

  addInPif (value String) -> checks whether the value is a token, constant, identifier or it does not belong in the language. If the item is a token, it gets added in the PIF. For a constant, it will get added in the PIF and the constants SymbolTable, for an identifier it will get added in the PIF and the identifiers SymbolTable. If the value cannot be put in any of those 3 categories, it will print an error message: Error at "value"

  scanProgram (file: File) -> receives a file as a parameter and analyses the contents of it. For each line of the file, it splits it's contents by a series of parameters (whitespace { } < > [ ] ( ) . ; : ,) and for each value it calls the addInPif method

  isToken (toCheck:String): Boolean -> checks if the toCheck String is a token

isConstant (toCheck: String): Boolean -> checks if the toCheck String is a constant

isIdentifier (toCheck: String): Boolean -> checks if the toCheck String is a identifier

toString(): String -> returns a string representation of the object