

**INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**TEORÍA COMPUTACIONAL**

**2CM4**

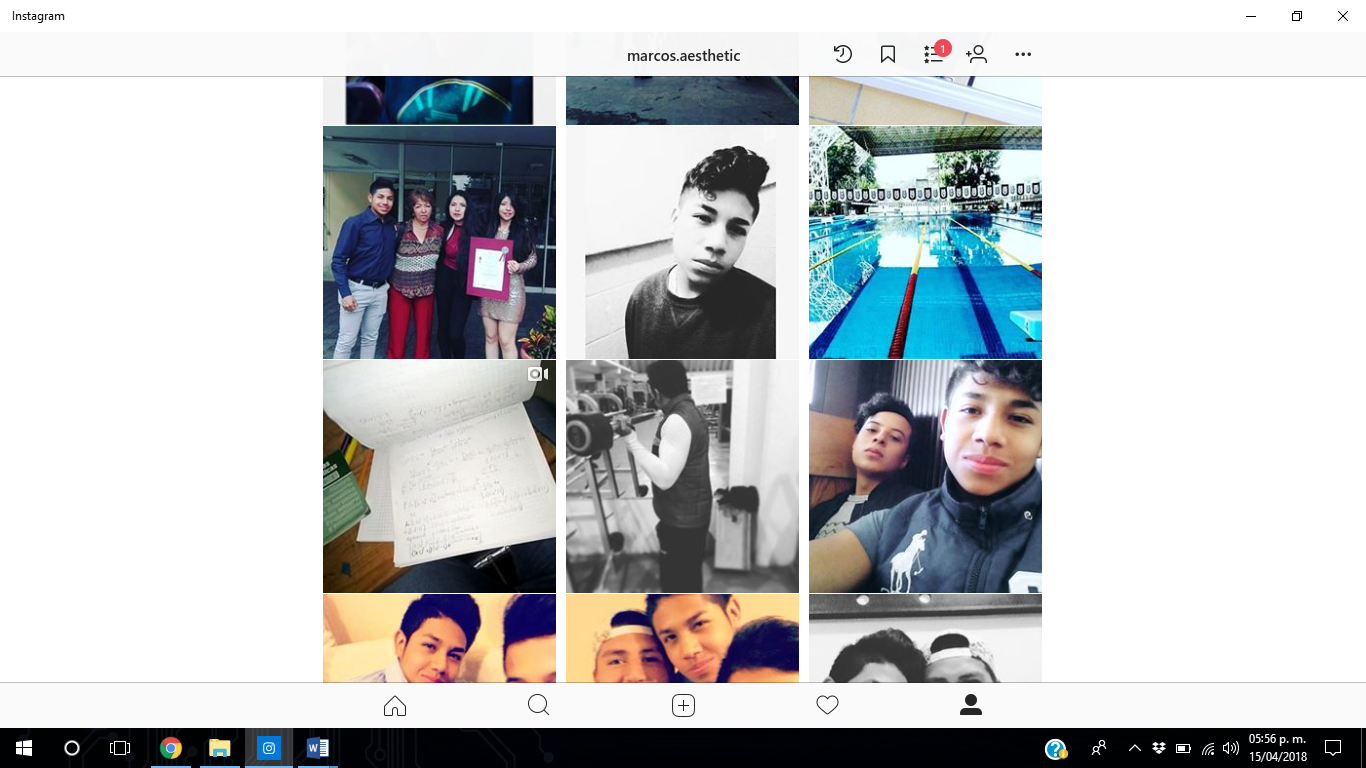
**PROFESOR: LUZ MARÍA SÁNCHEZ GARCÍA**

**PRÁCTICA 5 LIMPIEZA DE GLC**

**INTEGRANTES:**

**VÁZQUEZ MORENO MARCOS OSWALDO 2016601777**

**QUINTANA RUÍZ AJITZI RICARDO 2017631261**



**FECHA DE ENTREGA: 09 DE MAYO DE 2018**

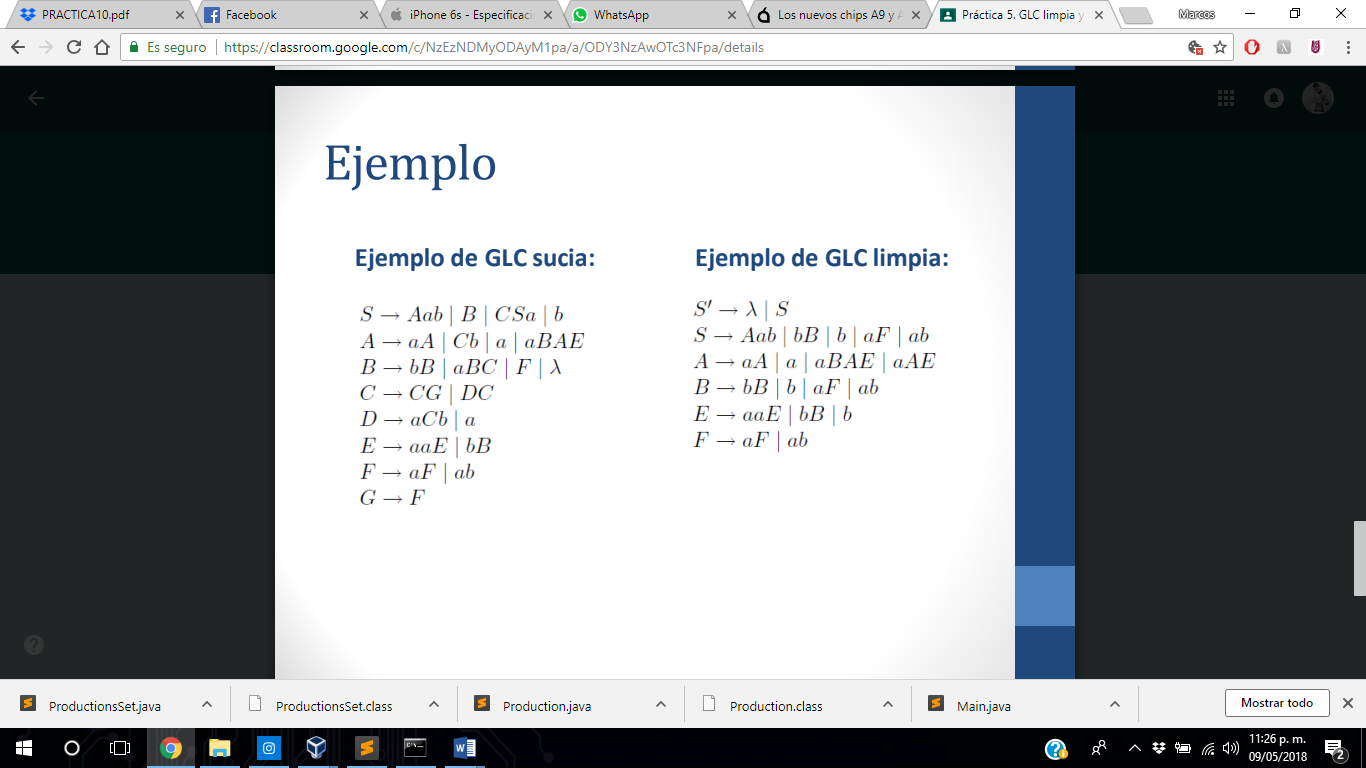
**INTRODUCCIÓN**

En la siguiente práctica se pretende realizar un programa en lenguaje de programación JAVA, el cual aceptará una Gramática Libre de Contexto (GLC) con sus reglas de producción leídas desde teclado y consola y realizará la limpieza de esta gramática en caso de estar sucia, obteniendo así una gramática limpia y bien formada, con tres casos principales:

* Muertas.
* Inaccesibles.
* Vacías.

**PLANTEAMIENTO DEL PROBLEMA**

Implementar el algoritmo de codificación de un programa el cual acepte una gramática libre de contexto desde un archivo o desde consola, con sus reglas de producción (en este caso pueden ser solo 3), limpiar la gramática y dejarla bien formada, a continuación, se tiene un ejemplo de una gramática sucia y a su lado una limpia y bien formada:



Teniendo de esta manera la eliminación de derivaciones con redenominación o redundantes, muertos o inútiles los cuales no tiene caso tenerlos porque de ninguna manera podemos acceder a ellos, también, los inaccesibles los cuales no pueden ser alcanzados porque en ninguna derivación de la gramática se tiene acceso y de esta manera estorban o ensucian la gramática.

**DISEÑO DE LA SOLUCIÓN**

Una vez con la problemática planteada se necesitó hacer tres distintos casos:

1. **El primero:** Cuando la gramática tenga solo un símbolo no terminal.
2. **El segundo:**  Cuando exista en la gramática dos símbolos no terminales.
3. **El tercero:**  Cuando la gramática contenga 3 términos no terminales.

Habiendo planteado estos tres casos, que en particular puedes hacer tan grandes tus gramáticas como se te haya pedido, en nuestro caso con solo 3 términos no terminales se nos fue permitido**,**  dicho lo anterior se plantean 4 funciones principales, las cuales se describen a continuación:

* **Validación de redundancia:**

Se toman la gramática ingresada como una cadena y se compara a no ser la misma en su producción o que en un ejemplo específico:

S->A (La raíz llama a A)

A-> aA | bB (A vuelve a llamar a aA y aB, en donde A pertenece a la raíz)

Lo que genera una redundancia o una llamada a si misma, por lo que lanza un error, deshaciéndonos de la raíz S y haciendo a A la nueva raíz.

* **Validación de producción muerta:**

La segunda validación es cuando la gramática se toma como una cadena la cual se compara con las cadenas anteriores insertadas como derivaciones guardadas en una cadena auxiliar y si no encuentra esa cadena dentro de la sub-cadena, lanza un error y en automático la limpias, como en el siguiente ejemplo:

S-> aA | bB

A-> aa | bB

B-> b | abC (En este caso no tenemos a C como una variable no terminal).

Por lo anterior se dice que C no debe exisitir en esta gramática para que la misma sea una gramática limpia y bien formada, eliminando a toda la cadena ingresa “abC”, quedando la gramática limpia de la siguiente manera.

S-> aA | bB

A-> aa | bB

B-> b

* **Validación de producción inaccesible:**

Esta validación aplica cuando dentro de las derivaciones de un símbolo no terminal no se llama a un símbolo terminal que puede usarse posteriormente, esto es de la siguiente manera:

S-> aA | bB (Se puede llamar a A o B y nunca a C)

A-> aa | bB (Se puede llamar a A o B y nunca a C)

B-> bB | b (Se puede llamar a B pero nunca a C)

C-> abc (A este símbolo no terminal nadie puede ingresar)

Por lo anterior, se dice que C no tiene cavidad dentro de esta grámatica siendo solo un fantasma dentro de la misma, pese a que se tiene derivación no se tiene acceso a ella.

En nuestra implementación se tiene que comparar a todos y cada uno de los símbolos no terminales como sub-cadenas dentro de las cadenas ingresadas como derivación de S, A y B. Esto fue algo que se nos facilitó demasiado al usar el lenguaje de programación Java ya que en C no encontramos una función que nos permitiera hacer eso de manera eficiente y desarrollarla no nos estaba facilitando la implementación.

* **Producciones vacías:**

La cuarta validación y quizá la más tediosa en cuanto a implementación es la producción vacía que trata cuando un símbolo no terminal derive a ε ya que cuando épsilon lleve a otras y otras derivaciones en gramáticas de más de 4 símbolos no terminales el programa debe de ser más extenso e ir haciendo copias en cadenas auxiliares e ir buscando el símbolo no terminal adecuado para terminar ahí, a lo anterior se explica con un ejemplo a continuación:

S-> aA | bB

A-> aa | bB

B-> bB | ε (Producción vacía)

Lo que lleva a darle el valor junto de la derivación dentro de la cadena de derivaciones en donde se encuentre quien la deriva, quedando de la siguiente manera:

S-> aA | bB

A-> aa | bB

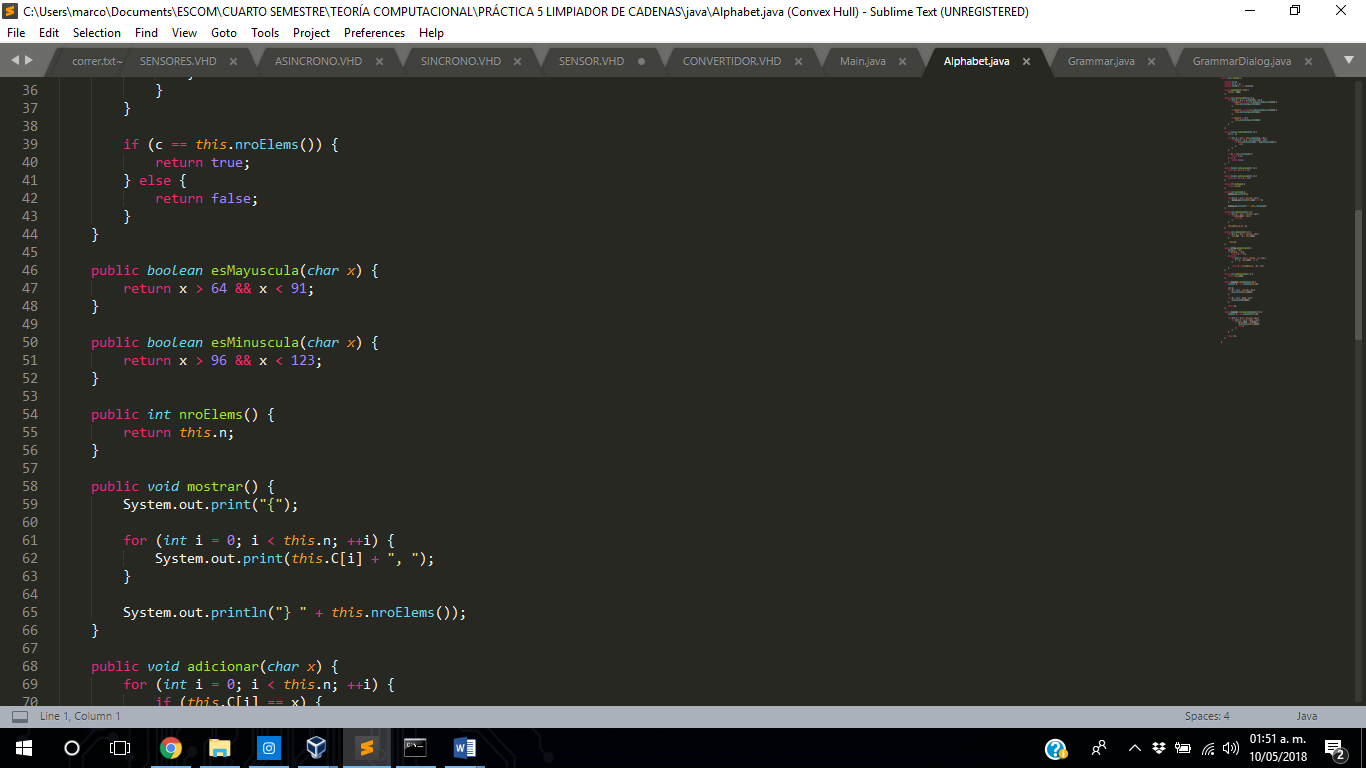
B-> bB | b

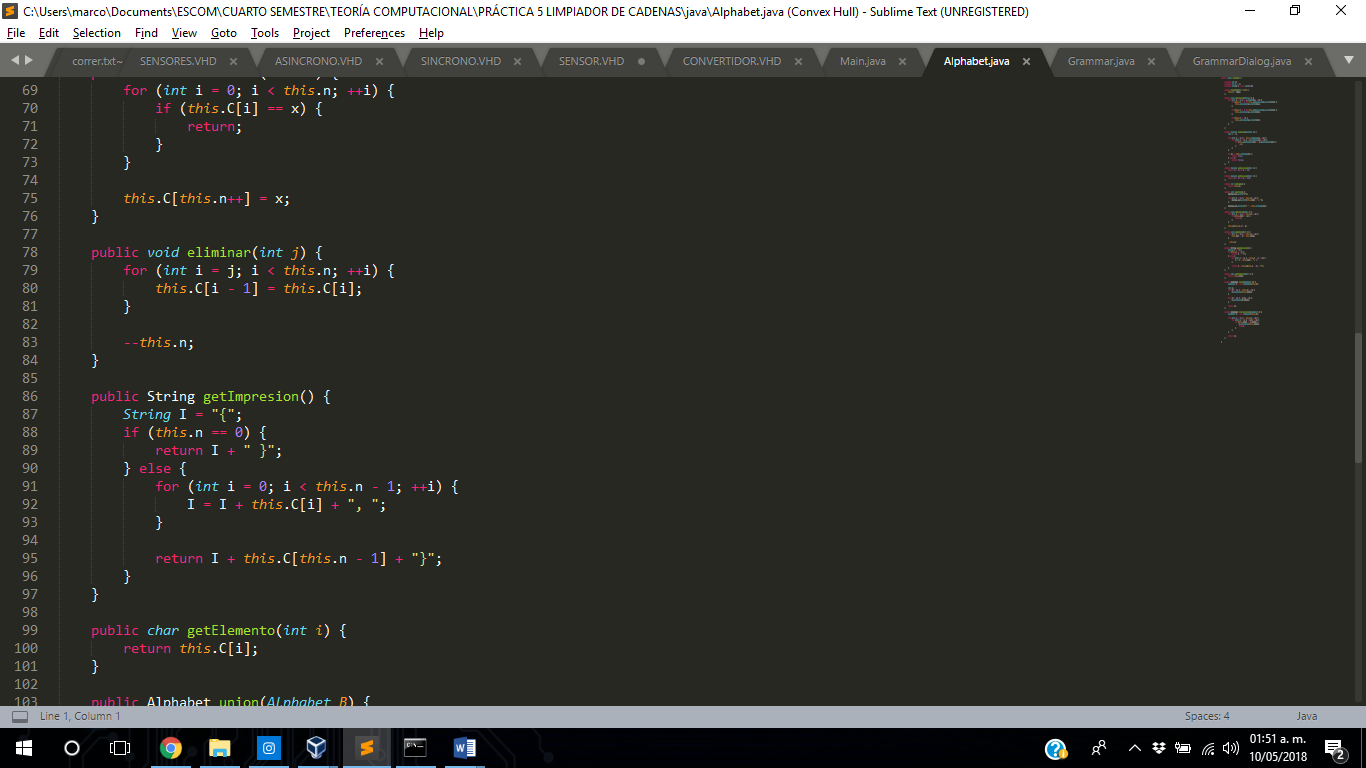
De B se deriva -> “bB | ε” y a épsilon le damos el valor de b por estar junto a B.

**IMPLEMENTACIÓN DE LA SOLUCIÓN**

**Alfabeto**

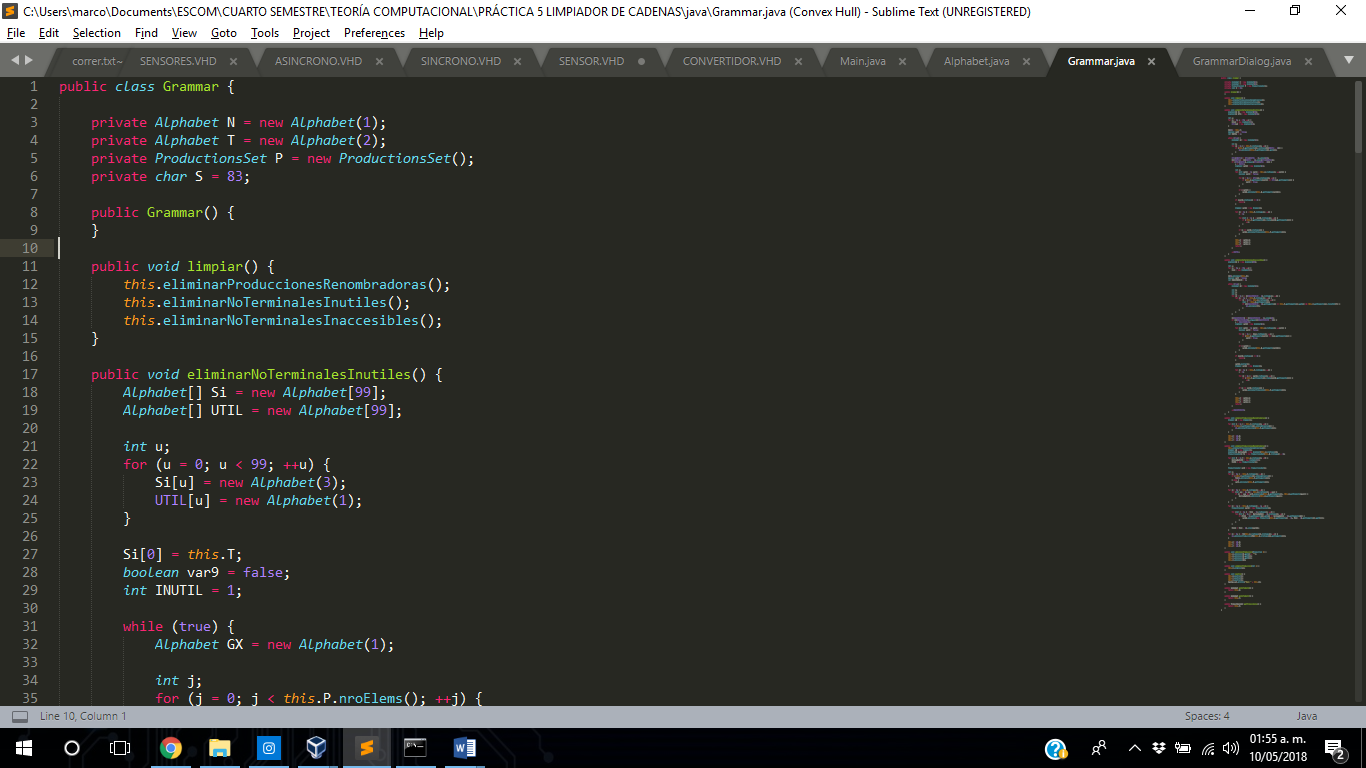


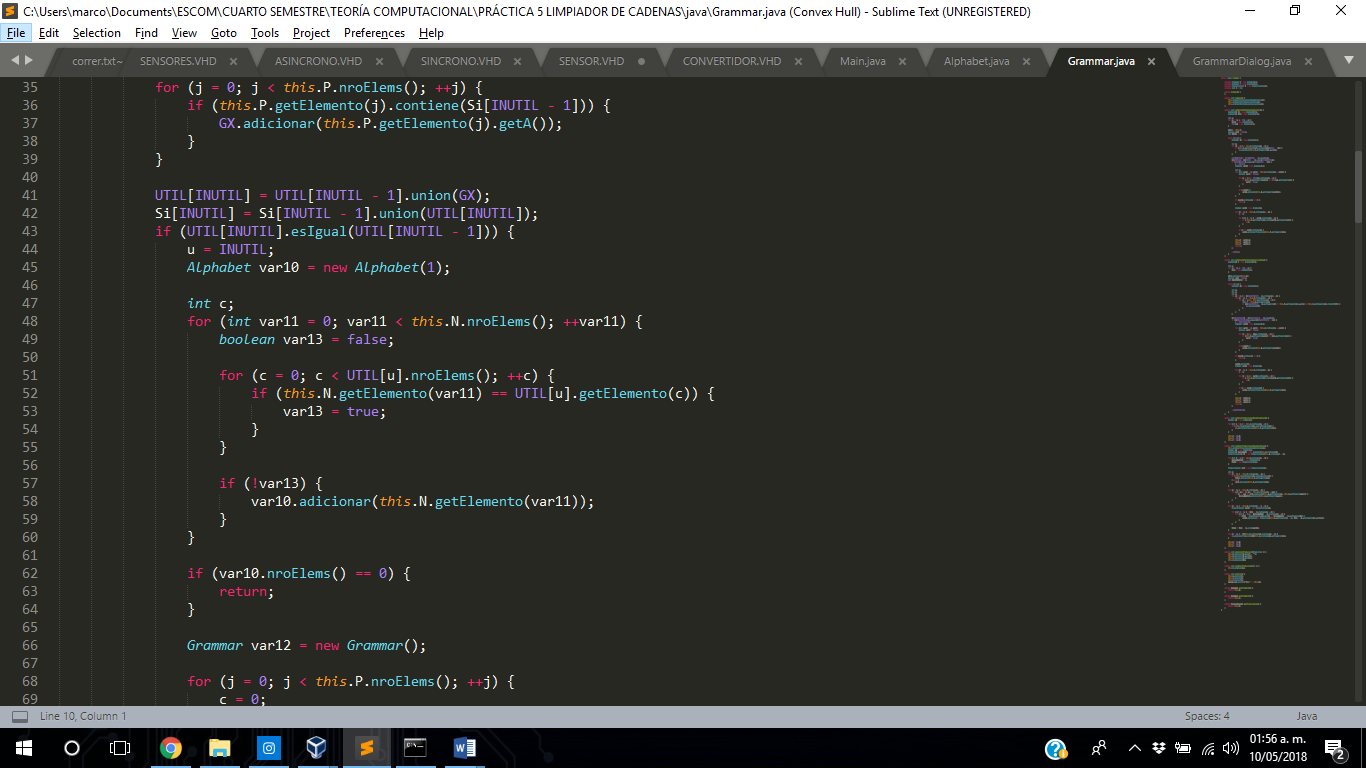


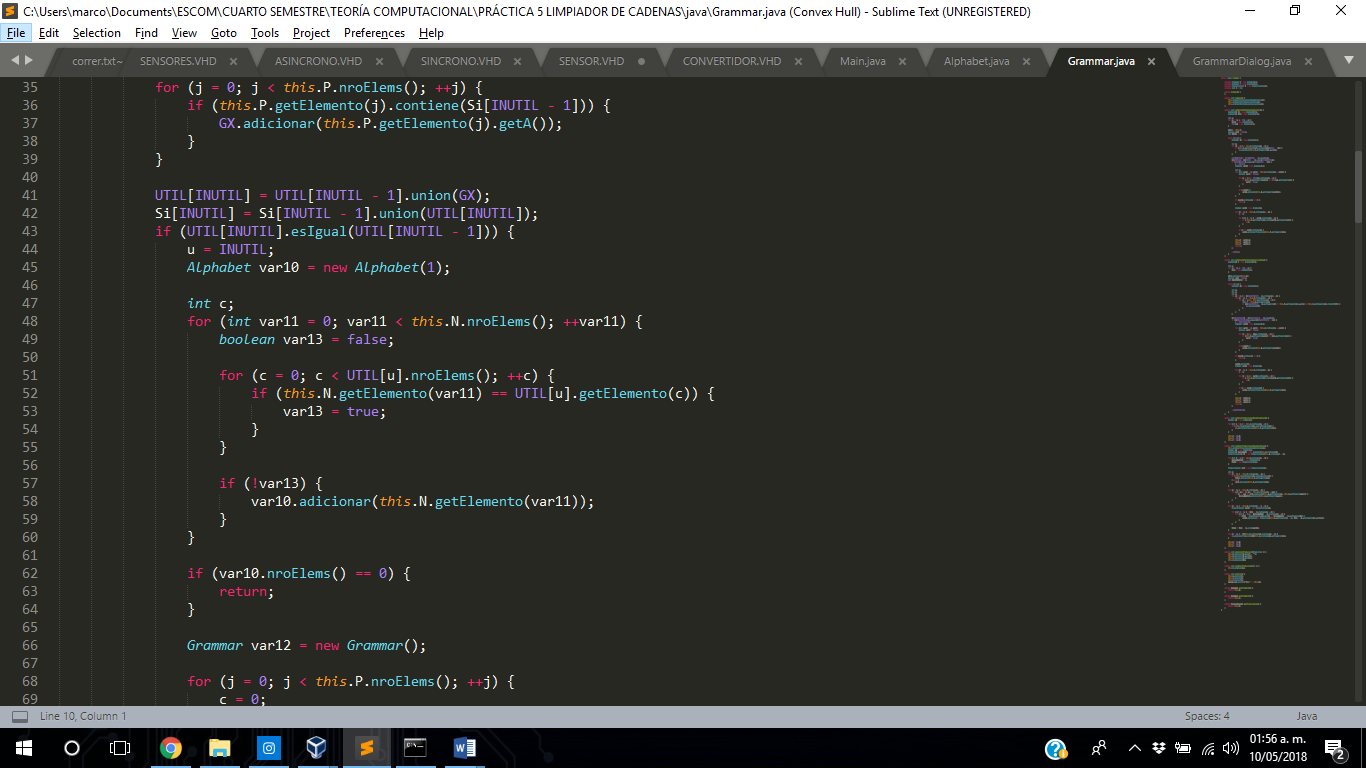


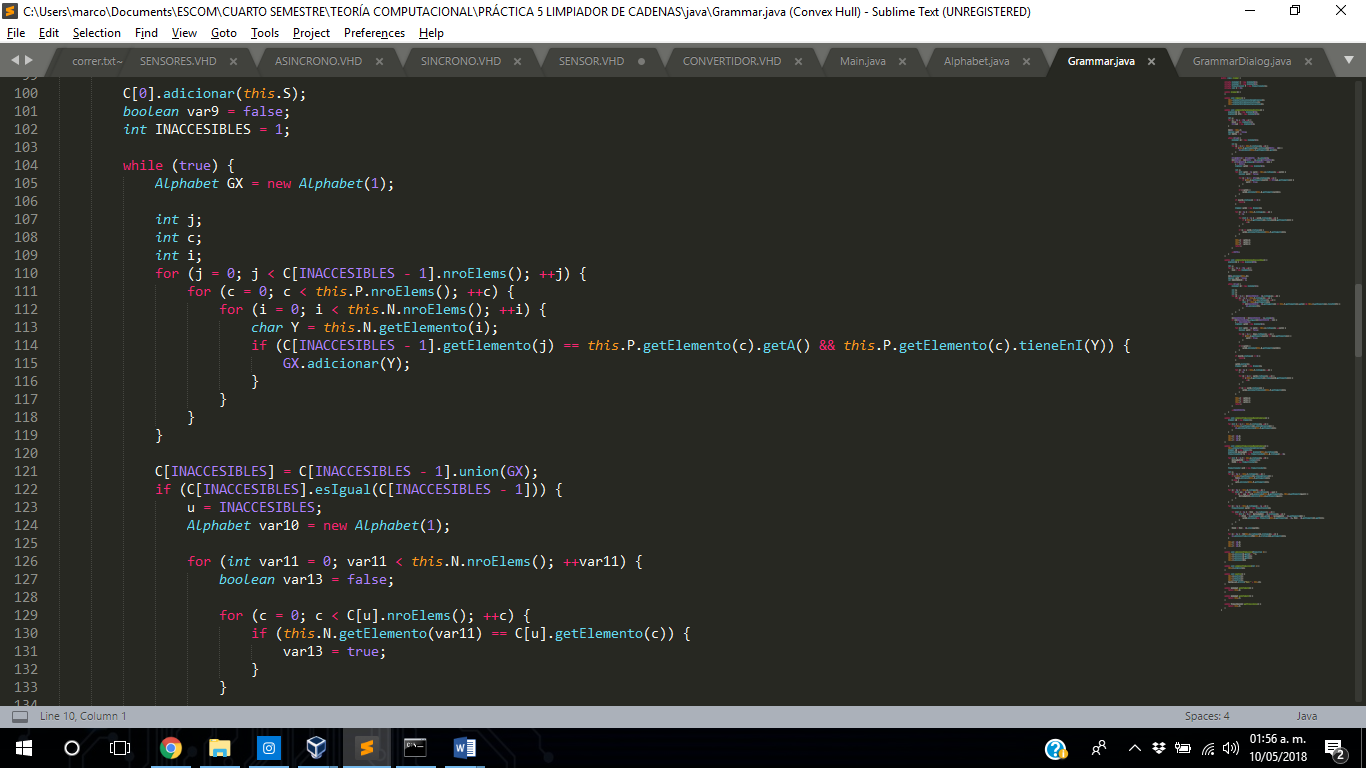


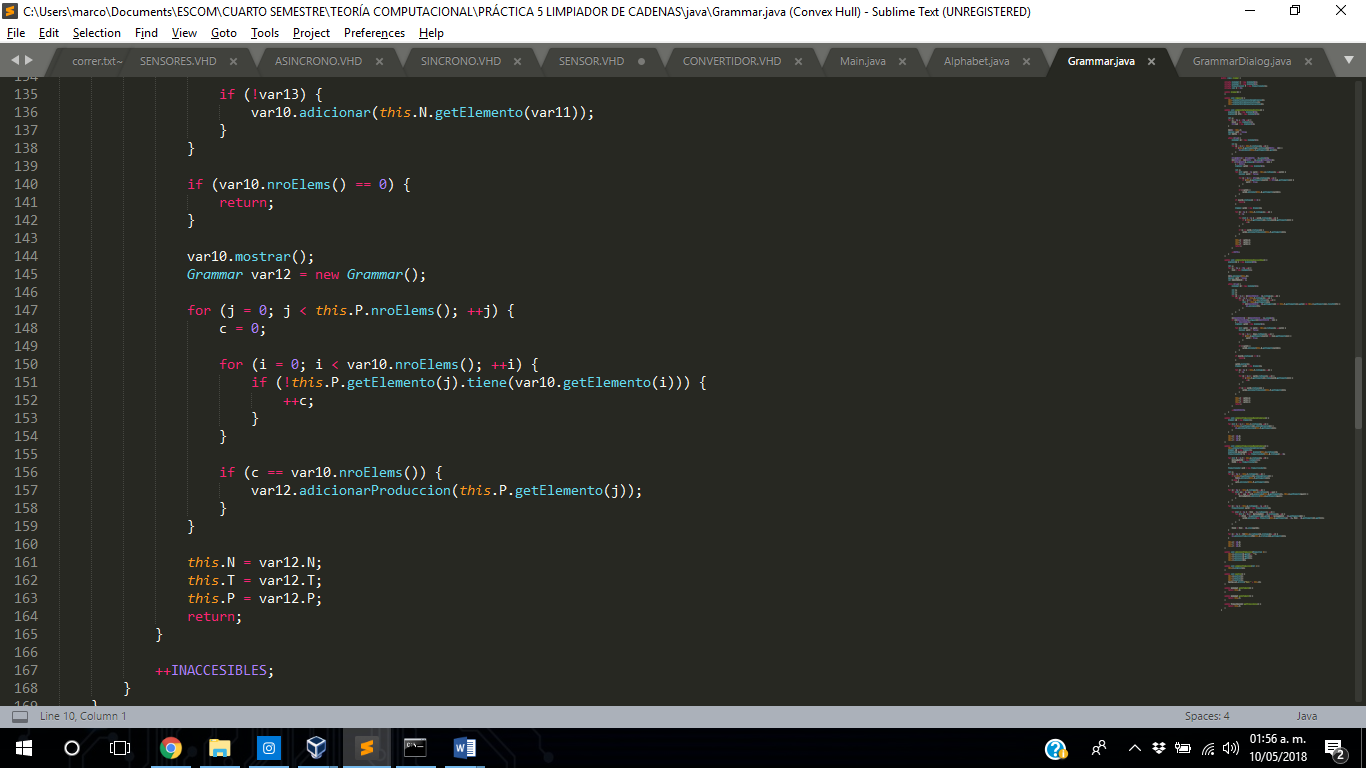
**Gramática**

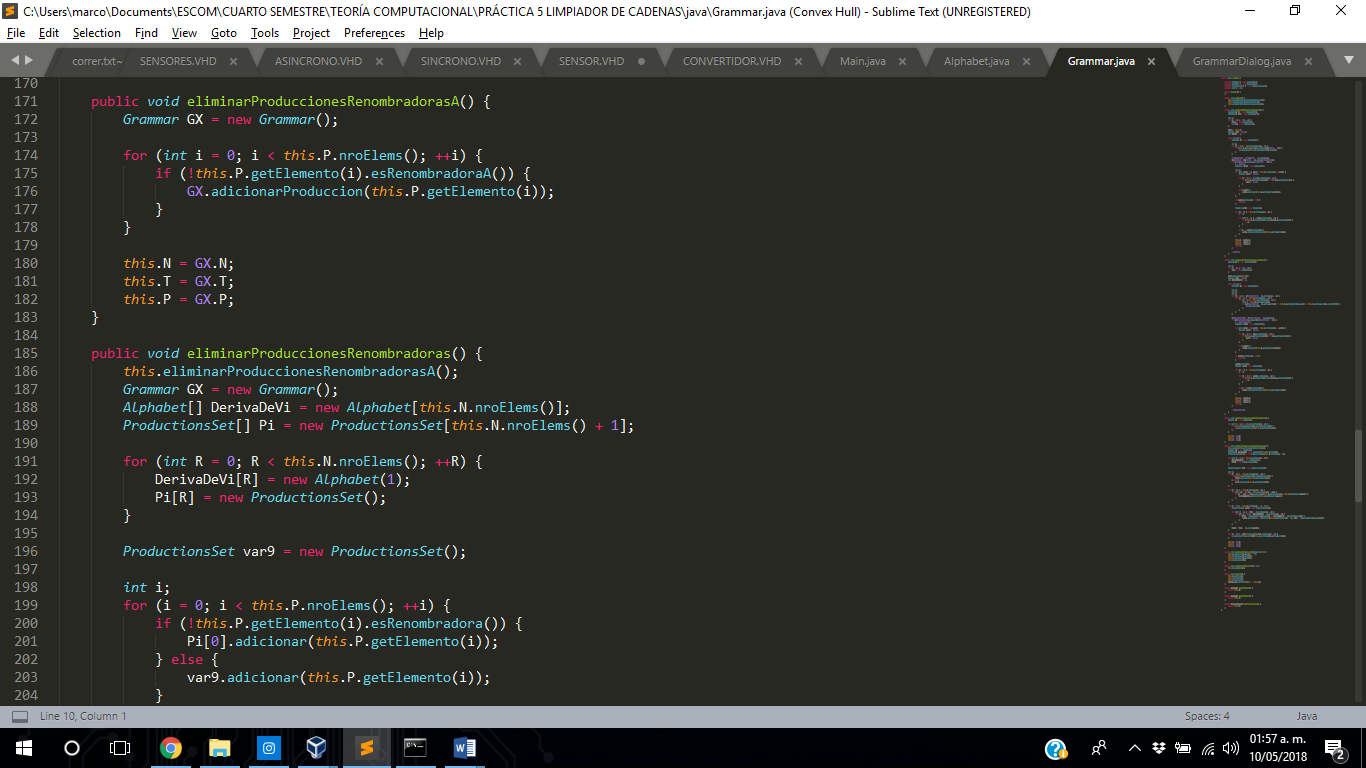




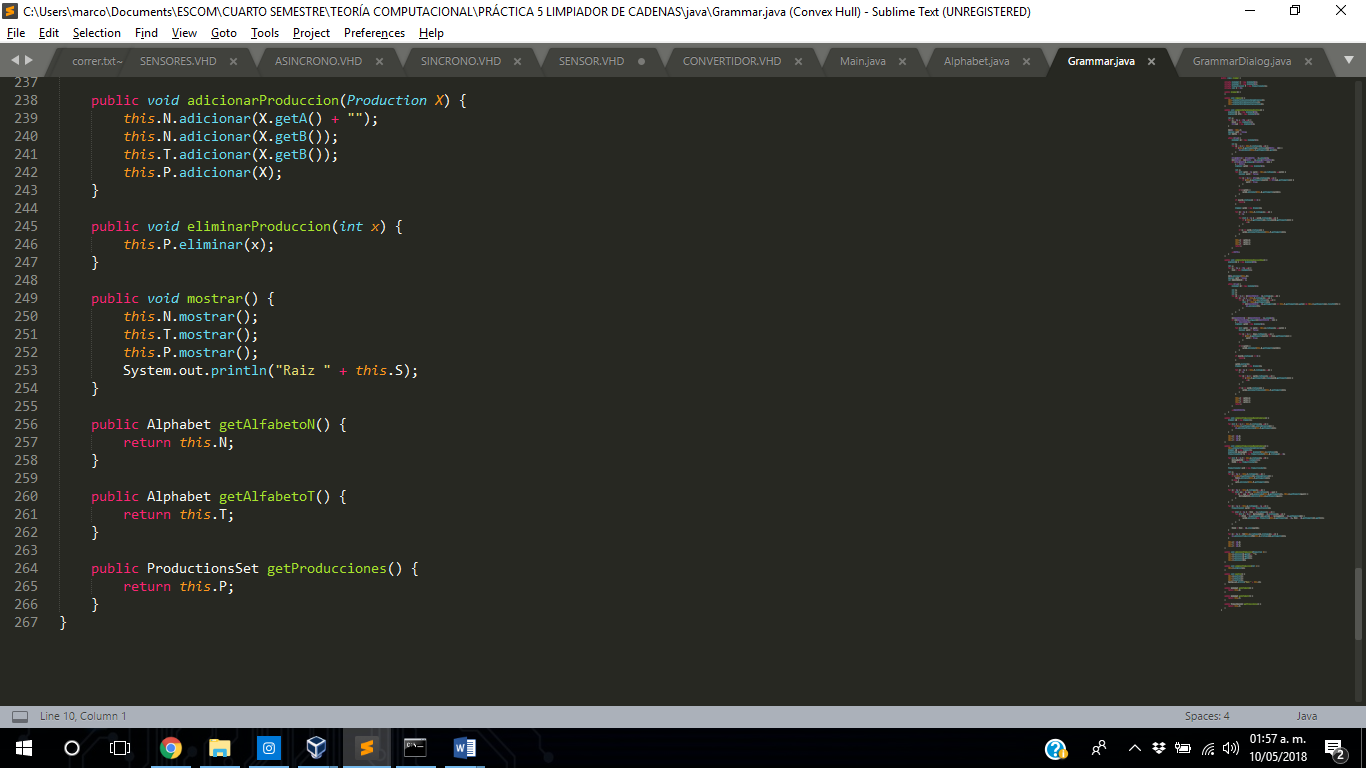






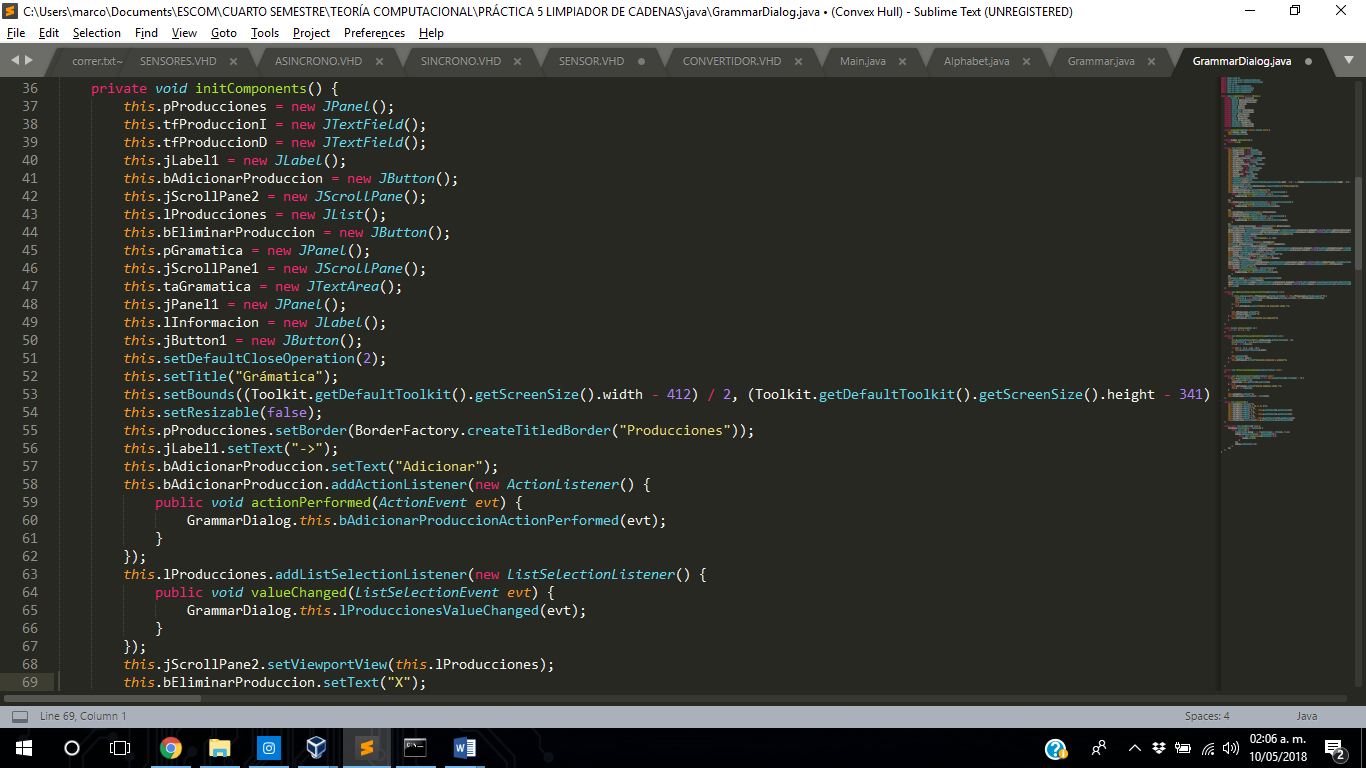


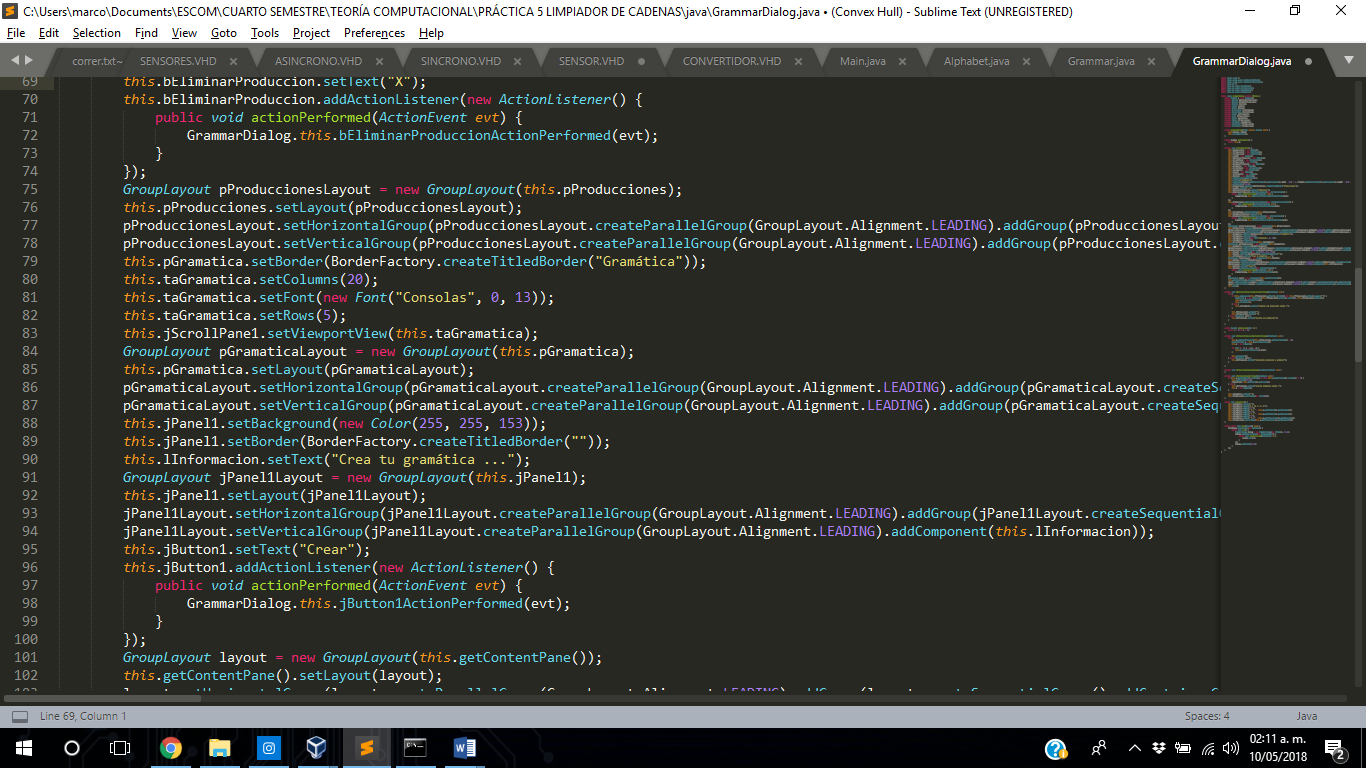


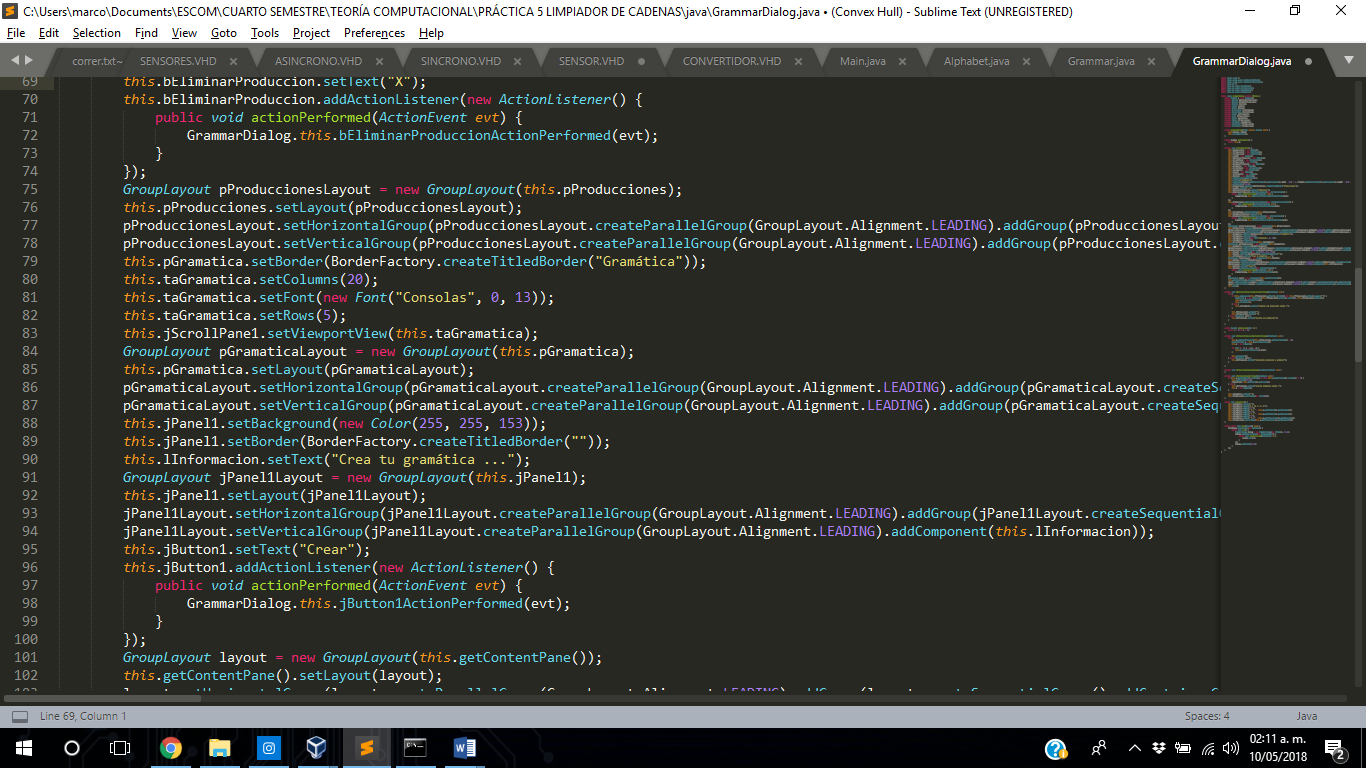


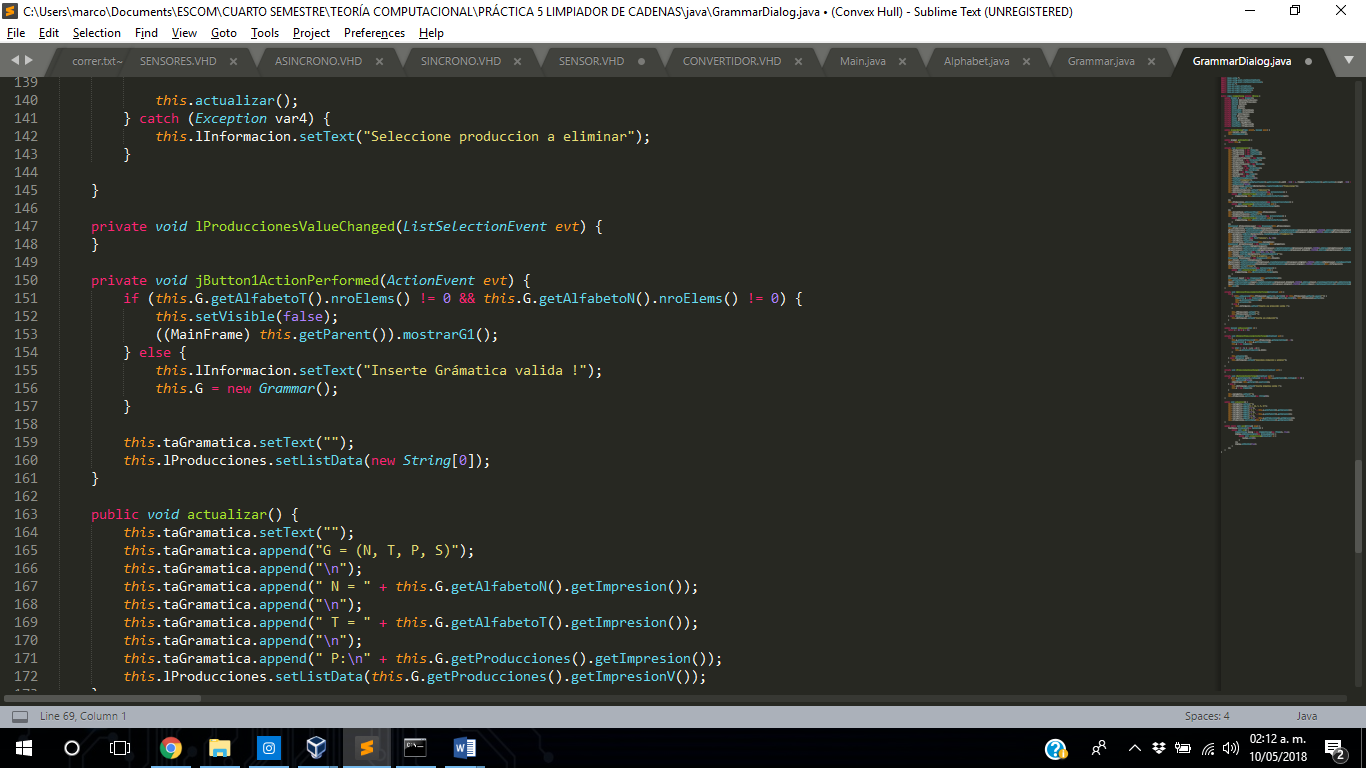
**Interfaz Gráfica**

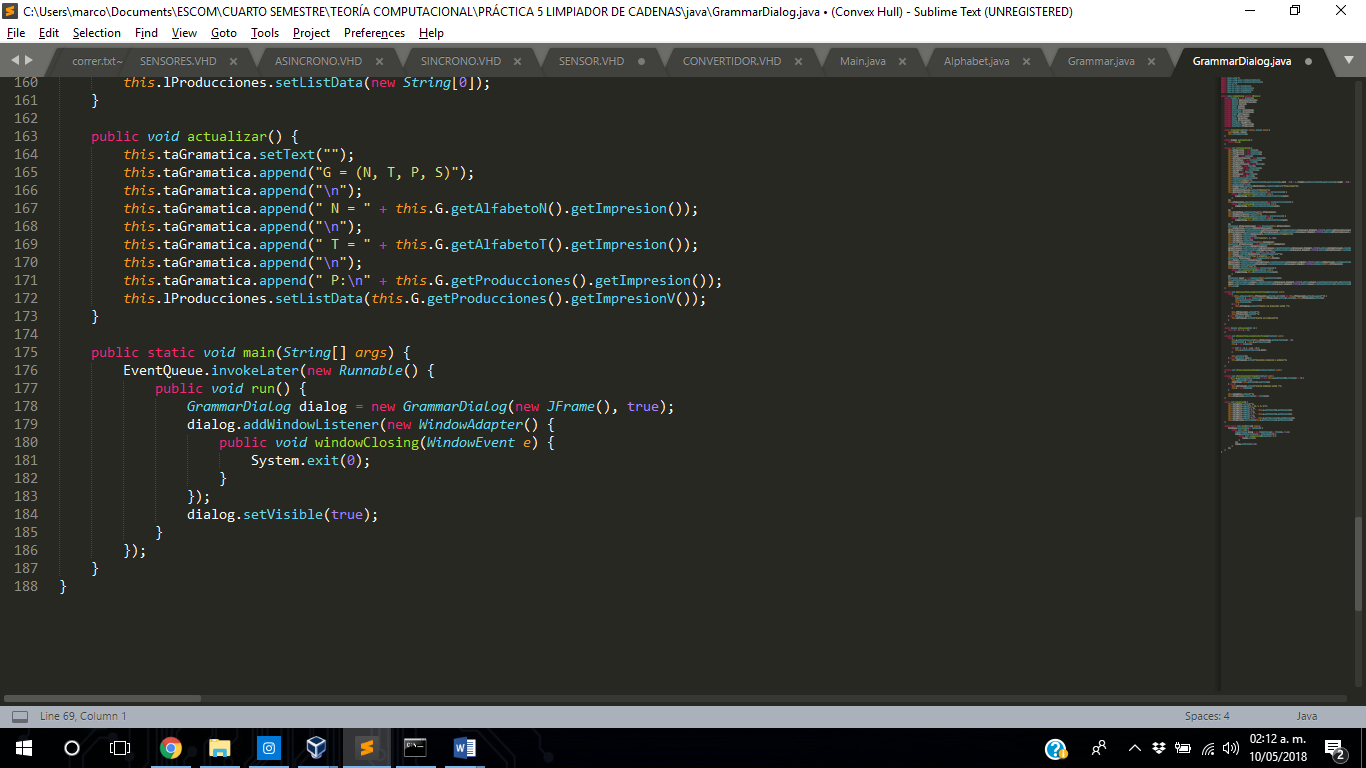




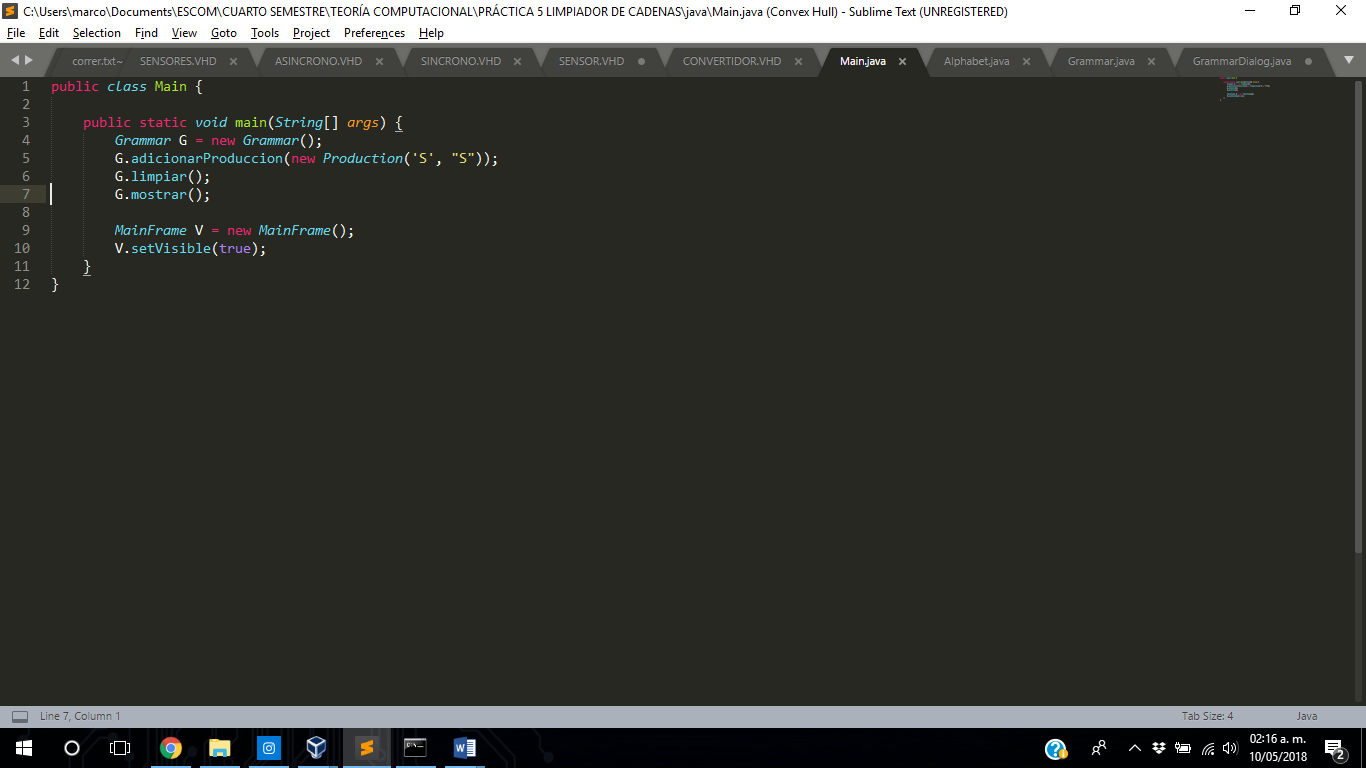




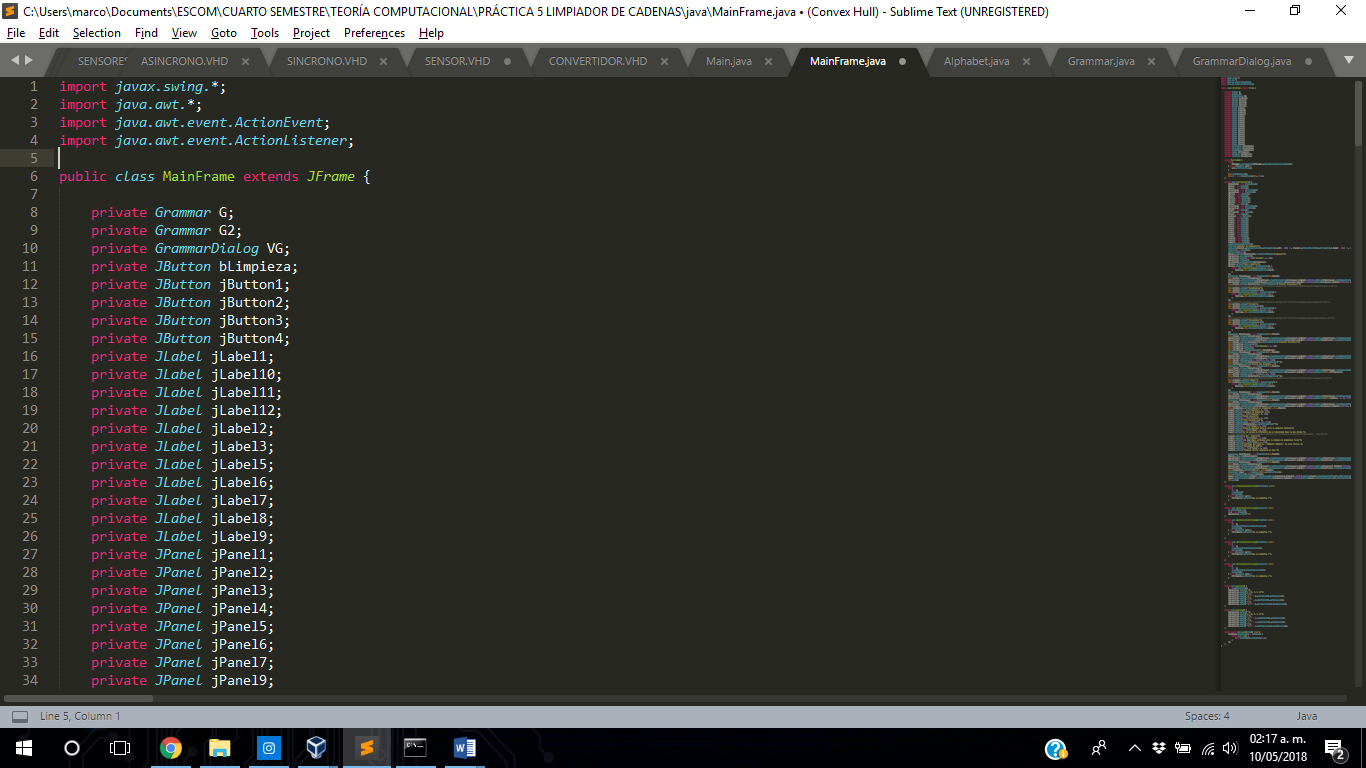


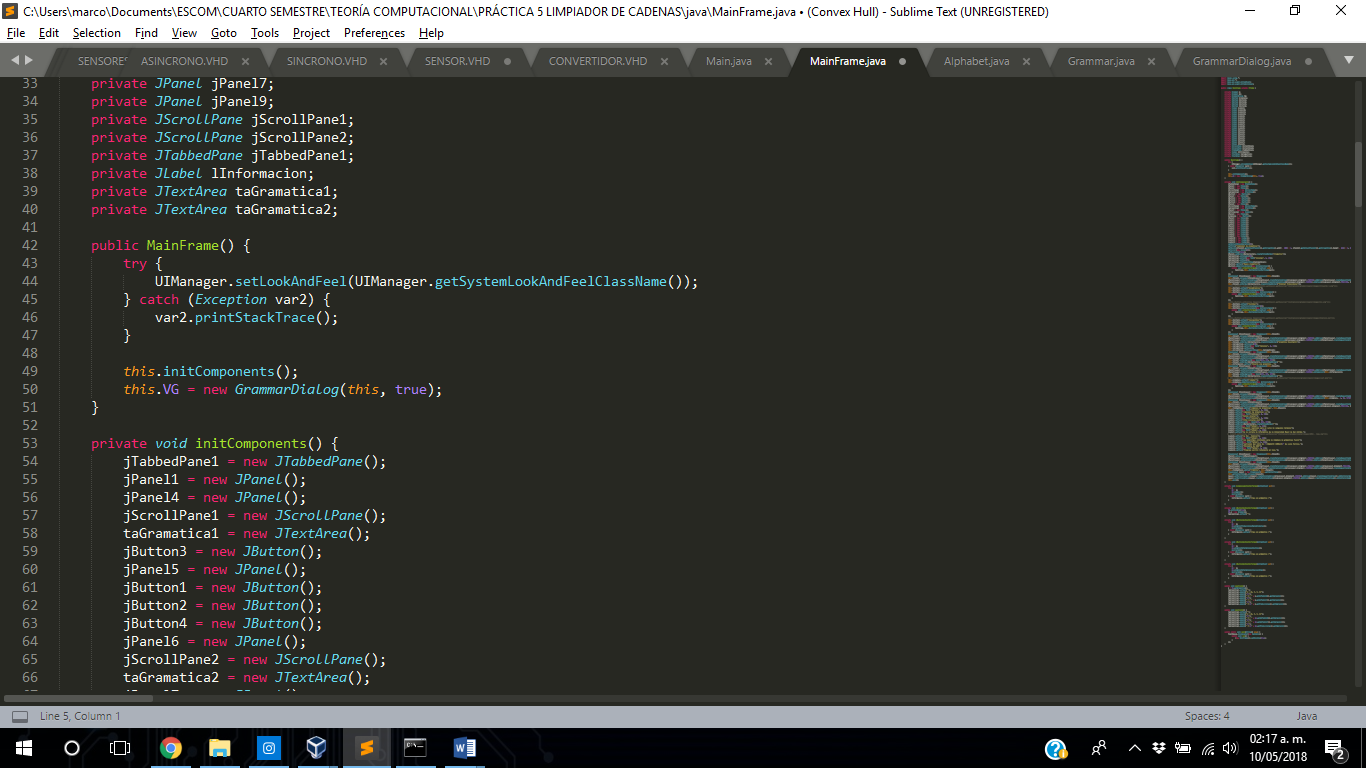


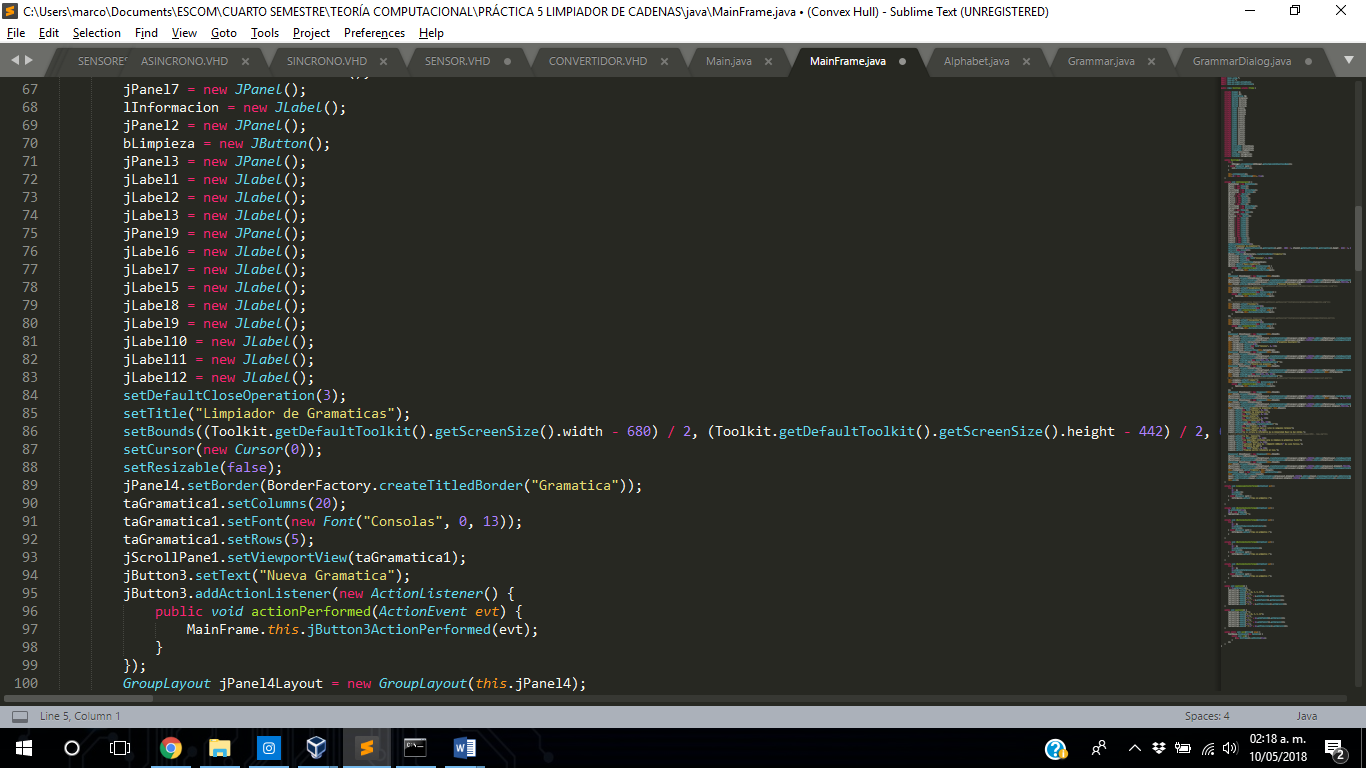
**Método main**

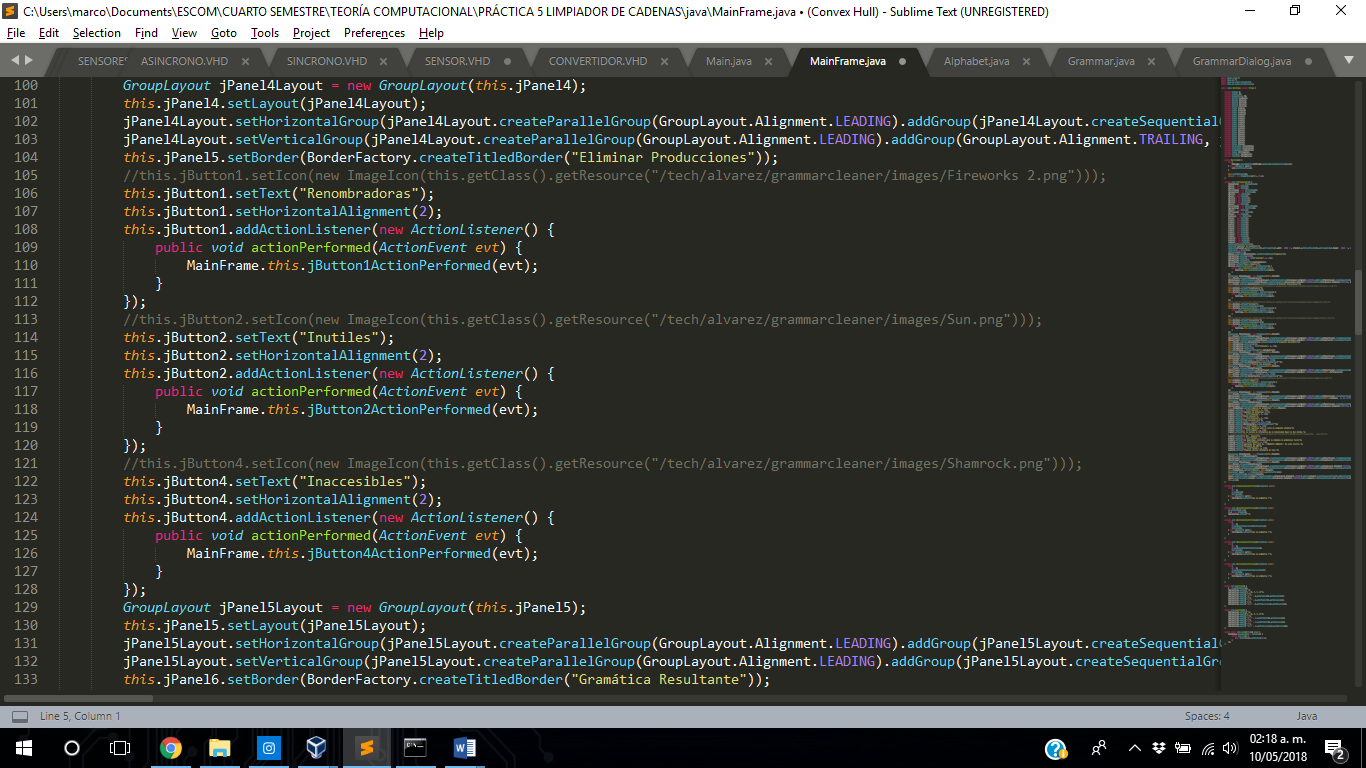


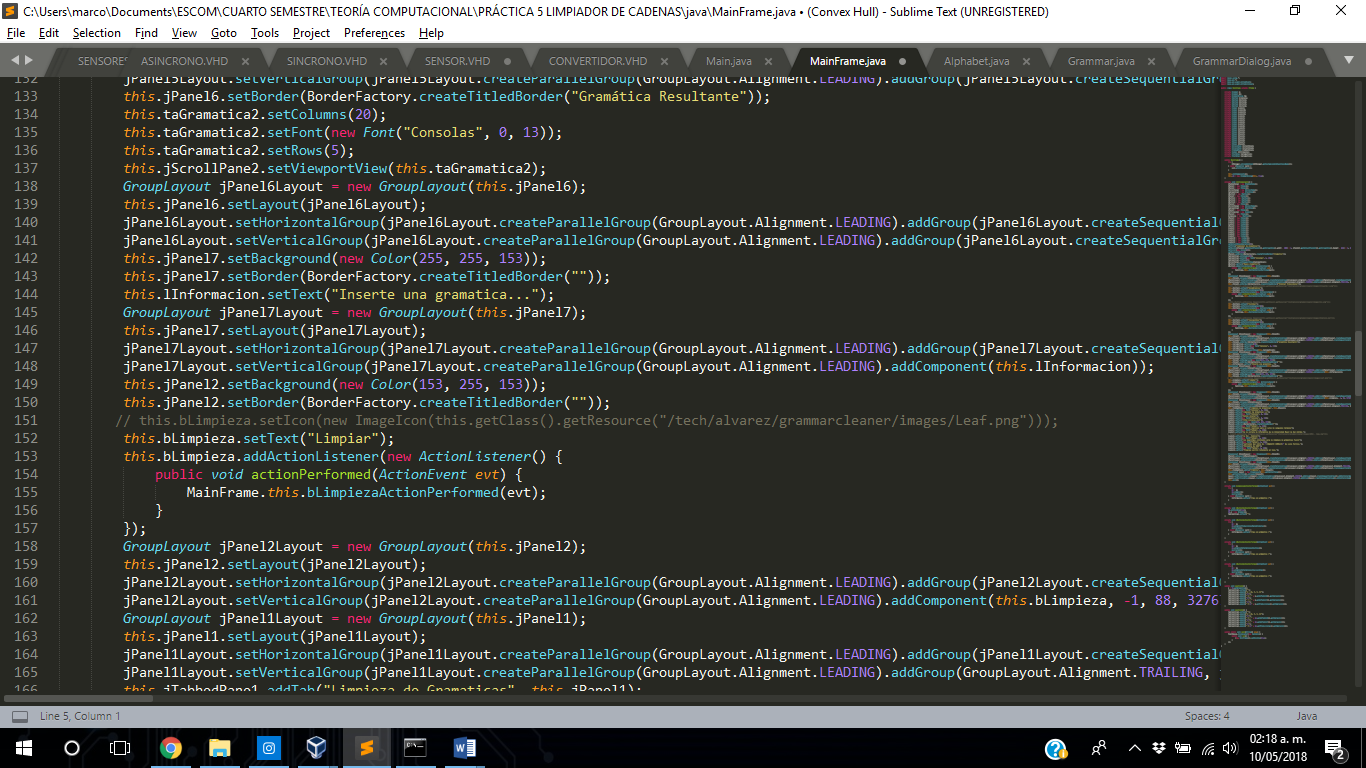
**JFrame Principal**

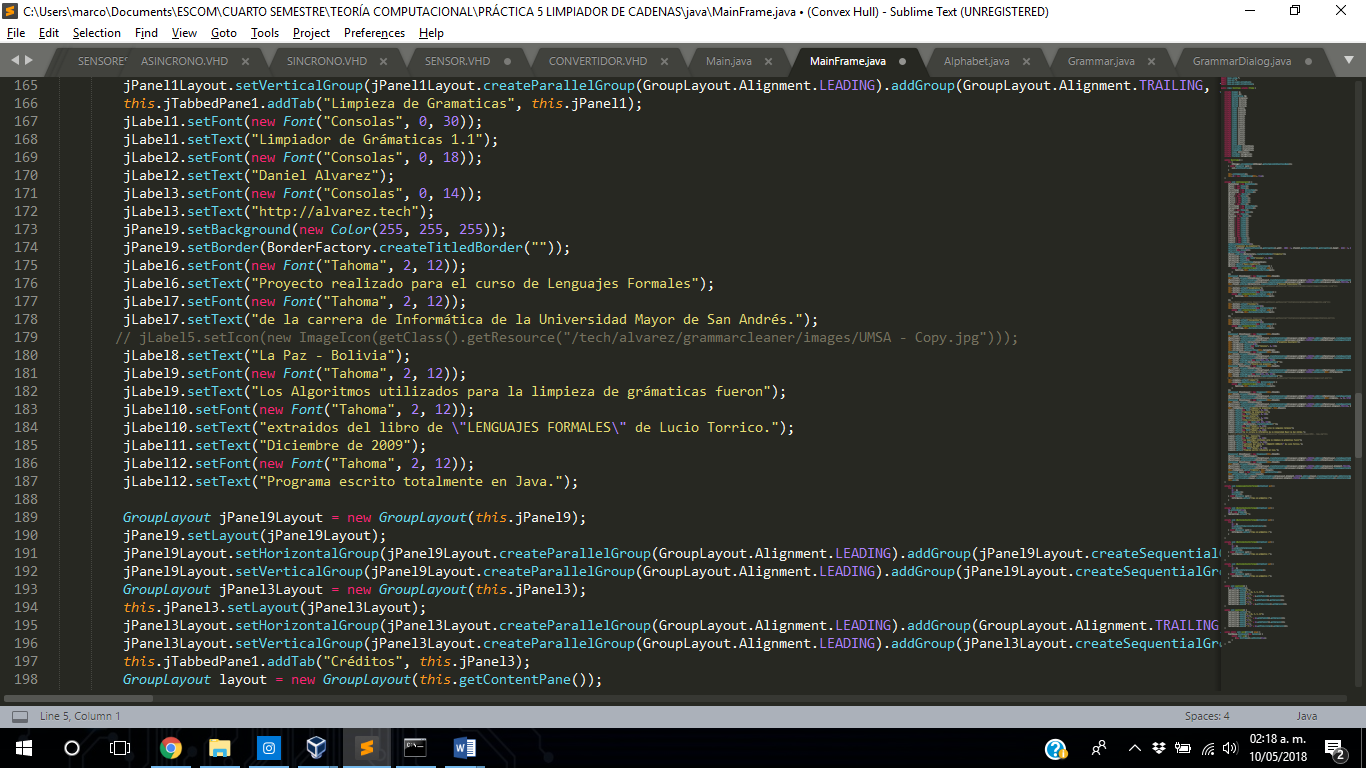






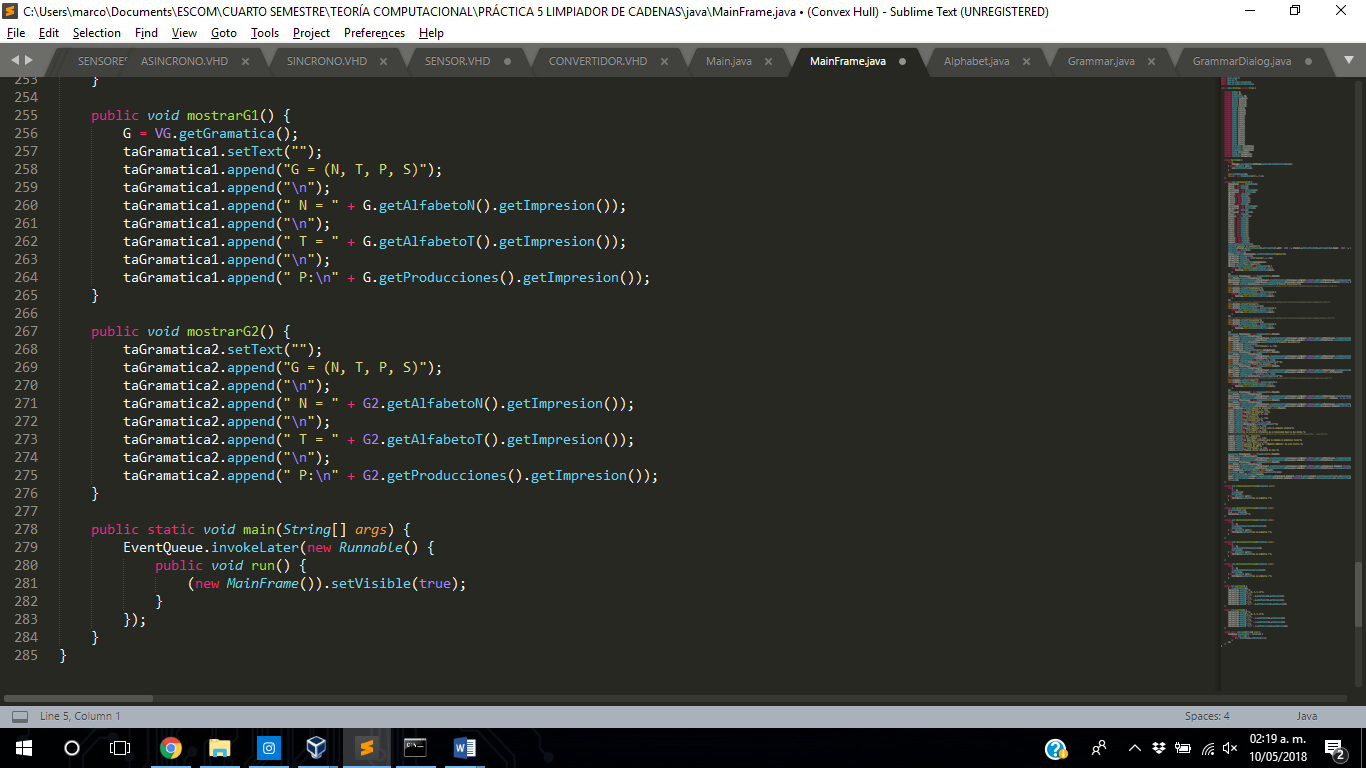




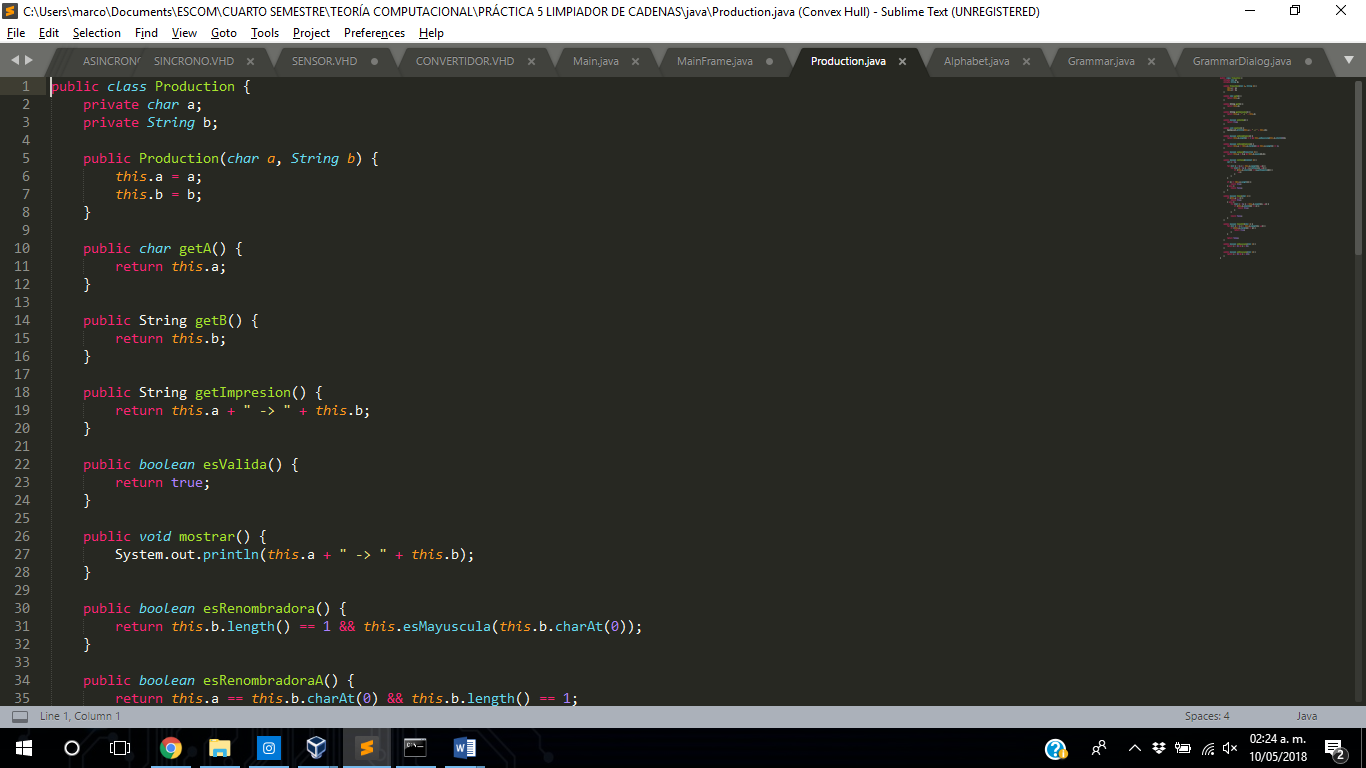


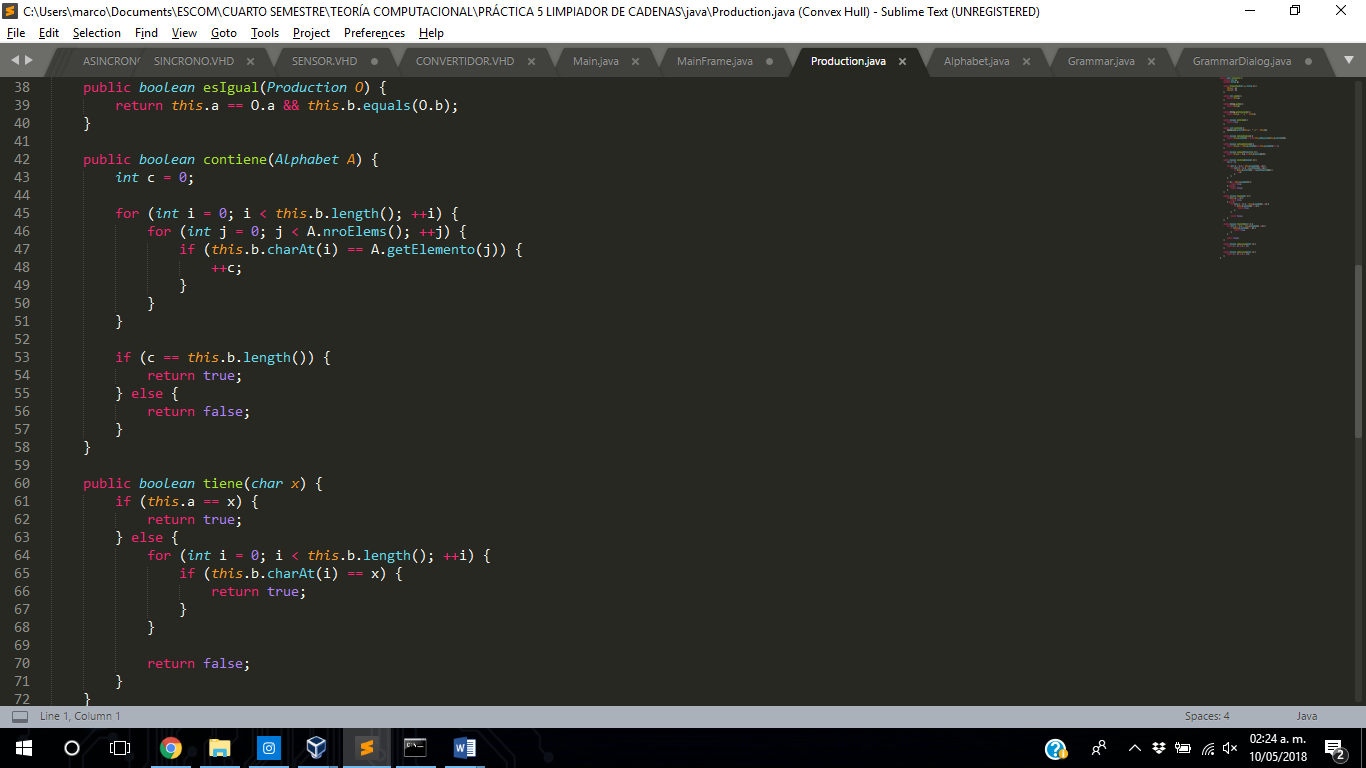


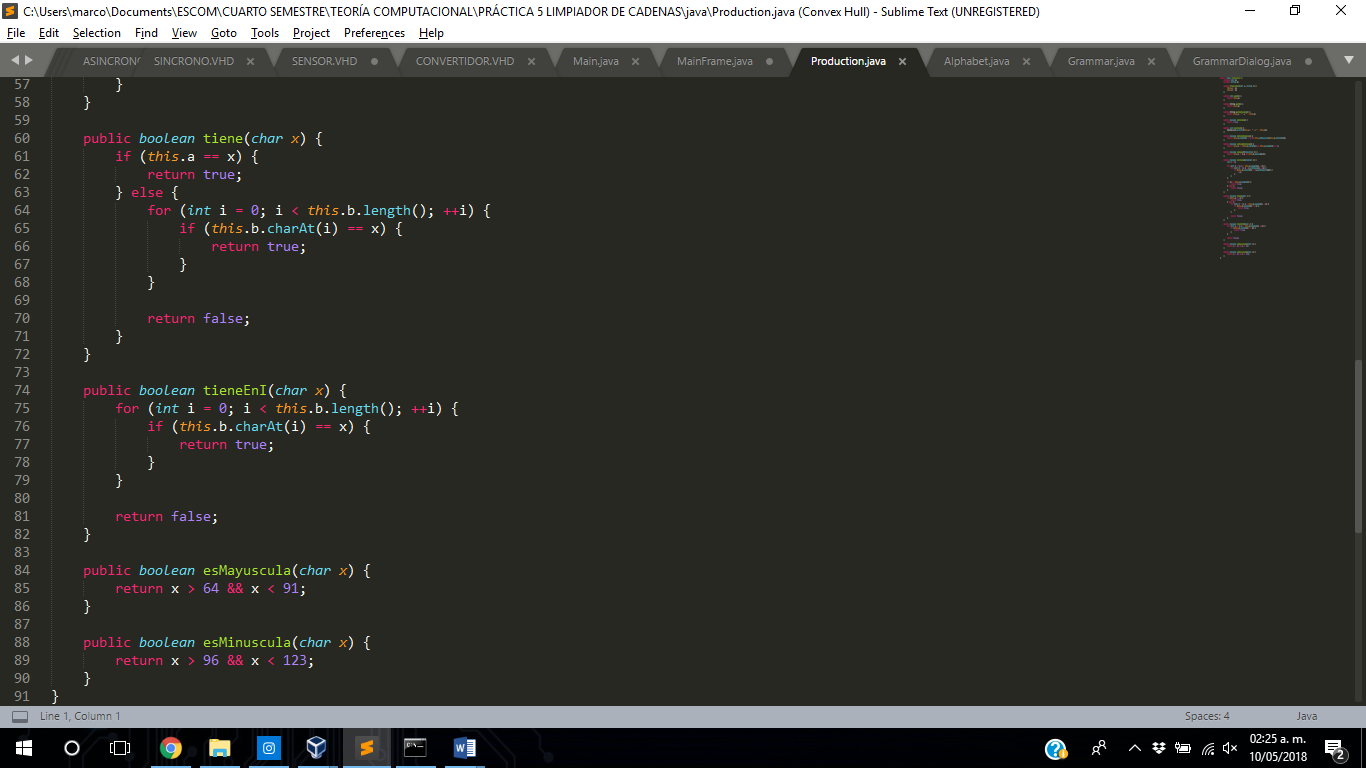




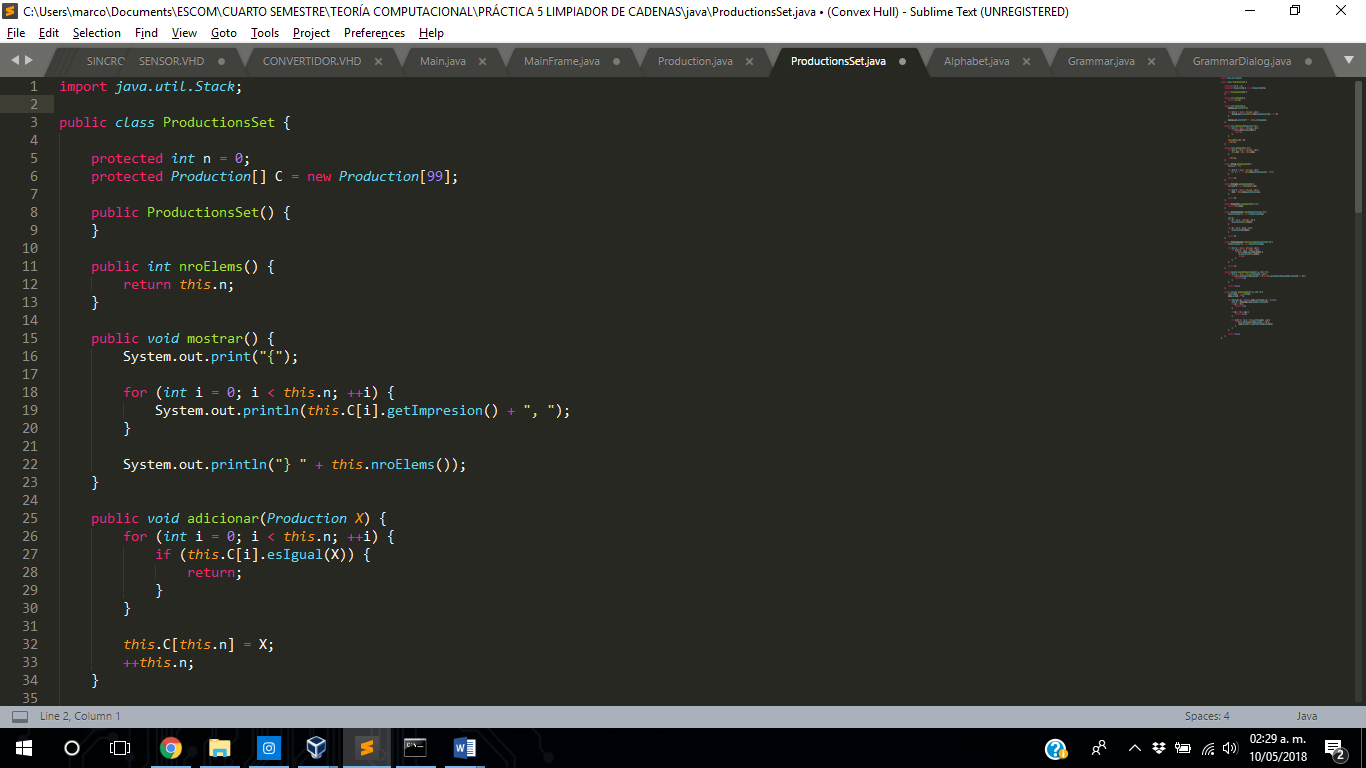
**Producciones**

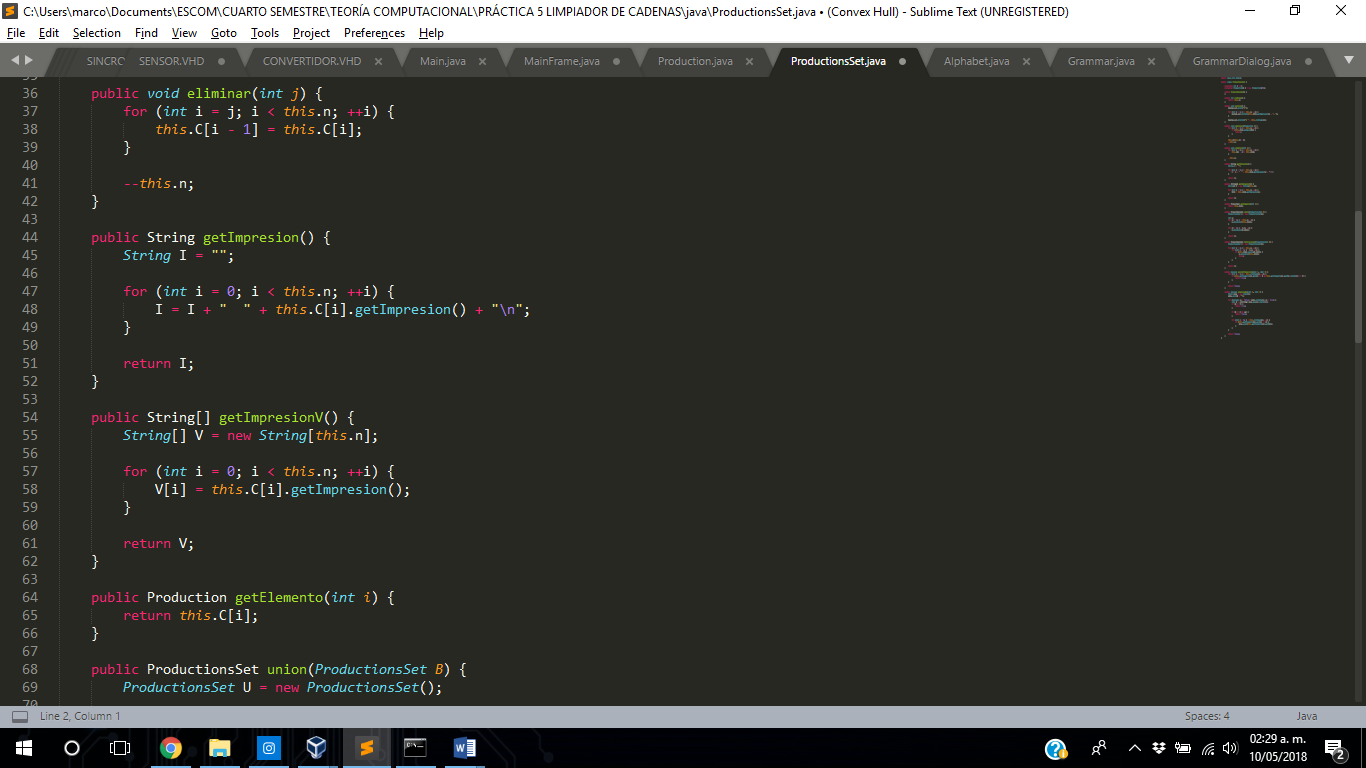




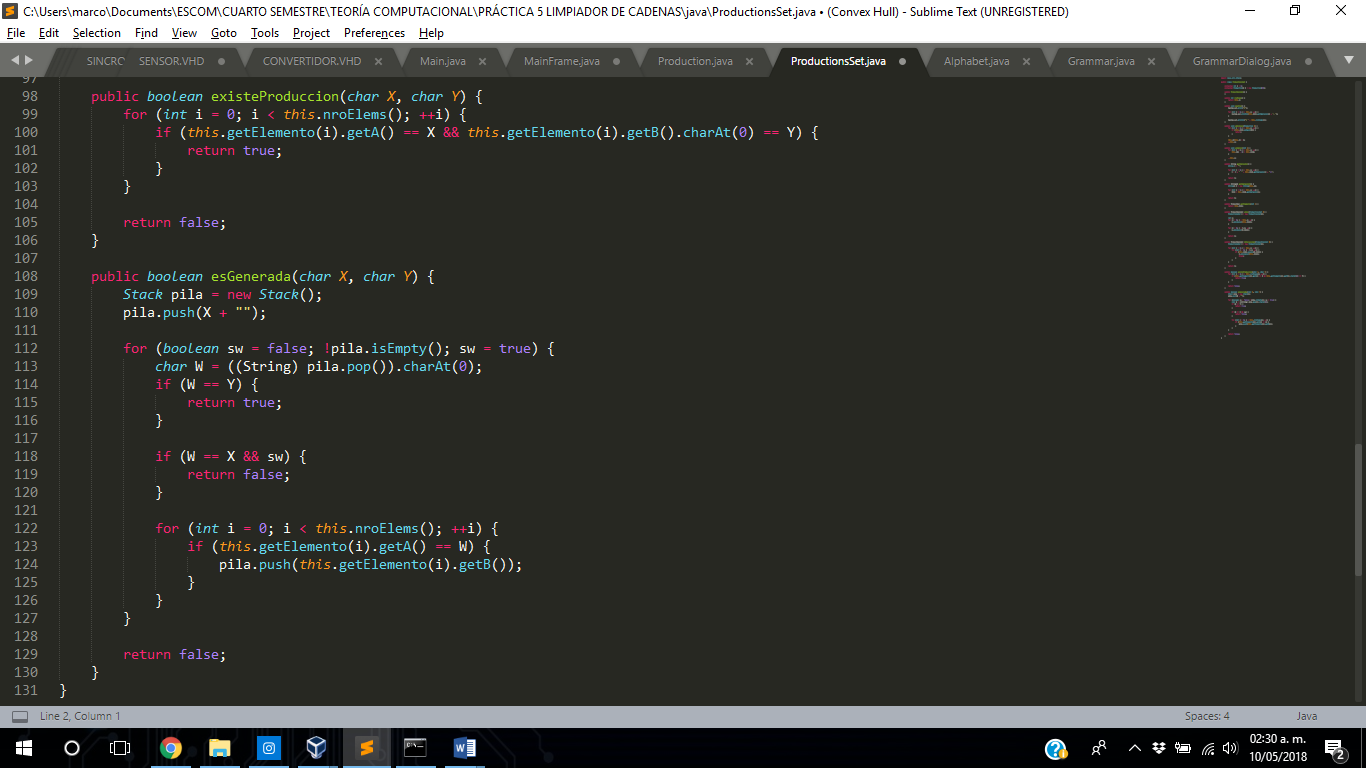


**Adiciones, Eliminaciones, Uniones e Intersecciones de elementos**









**FUNCIONAMIENTO**

**CONCLUSIONES**

Dada la complejidad de la práctica concluimos que sin duda fue una práctica de gran nivel, es una práctica que nos hizo esforzarnos al 1000 por cierto, puesto que había muchas cosas que no dominábamos del lenguaje de programación java y debimos buscar, investigar e indagar para lograr sacar adelante el objetivo de la misma, por otro lado mencionar que tuvimos que cambiar al lenguaje Java porque se intentó en C pero se complicaba demasiado el hecho de controlar las funciones de re dominación y accesibilidad, dada la situación no nos dimos por vencidos y nos quedamos con un grato sabor de boca al lograr que nuestro programa saliera a flote sin problemas, al tener las ideas claras acerca del programa nos dimos cuenta que no existía opción y el segundo lenguaje que mejor dominábamos era Java. Por lo anterior, nos llenamos de gratitud con la profesora y la unidad de aprendizaje porque sin duda nos está convirtiendo en unos alumnos dominadores no solo de un lenguaje sino de 2 y claro lograr aún más lenguajes.

Se logro el planteamiento del problema con éxito y un buen trabajo en equipo.