MANUAL DE USUARIO

Requerimientos

Sistema Operativo (Linux)

Node js, Angular

Entender los lenguajes JAVA (para editar archivos en el IDE).

Los archivos deben de tener la extensión: .java

Descripción

El programa es un editro de código, el cual genera código de tres direcciones a partir del archivo de entrada que se este leyendo.

Opciones Disponibles

Nuevo proyecto

El IDE debe permitir la creación de proyectos donde se organizan los archivos con el código a compilar. Esta opción permite al usuario crear un proyecto nuevo que será almacenado en el servidor usando una estructura xml.

Nuevo archivo

Permite agregar un archivo nuevo al proyecto, se debe indicar el paquete al que se agrega el archivo.

Abrir proyecto

Esta opción permite abrir un proyecto almacenado en el servidor. La interfaz de usuario debe mostrar el proyecto usando una estructura de árbol donde se muestran los paquetes y archivos del proyecto.

Abrir archivo

Usando el área de despliegue de proyecto, el usuario puede seleccionar un archivo .java para editarlo, éste contiene el código fuente de un programa almacenado con anterioridad.

Guardar

Guarda el contenido de la pestaña activa en ese momento.

Guardar Como

Permite guardar el contenido de la pestaña activa en ese momento, con diferente nombre en algún paquete del proyecto.

Cerrar proyecto

Cierra el proyecto del IDE.

Código en Tres Direcciones

Esta opción genera el código en tres direcciones del proyecto y este resultado deberá ser mostrado en la pestaña Código Tres Direcciones.

Código Assembler

Esta opción genera el código Assembler del programa y se mostrará el código generado en la pestaña Assembler

Definición del Lenguaje

El compilador trabaja sobre un lenguaje de programación con sintaxis java, es decir que la sintaxis, la jerarquía de operaciones aritméticas, lógicas, etc. y aquellas características que conformen el lenguaje para el compilador, serán las mismas que en el lenguaje de programación java, exceptuando los siguientes casos:

- Los tipos de datos primitivos a manejar son: float, int, char, boolean y string.
- La precedencia de operadores y la conversión implícita de los tipos de datos es la misma que en java.
- Las expresiones aritméticas serán de forma limitada: +, -, *, /, %, además de las definidas en los siguientes puntos.
- Operaciones de la clase math:
    - Math.abs
    - Math.ceil, Math.floor, Math.round
    - Math.max, Math.min
    - Math.pow, Math.sqrt
    - Math.random
    - Math.toRadians
    - Math.acos, Math.sin, Math.atan, Math.exp
- Incremento y decremento (i++, i--), (No se incluyen l
- Todas las clases pueden incluir
    - Modificador de acceso (únicamente se trabajará el caso public, por lo tanto se puede incluir public class Class1 o class Class1, ambos casos son de acceso público).
    - Las clases pueden incluir o no el keyword final. (public final class Class1 o final class Class1).

- Existirá un método void main (cómo la versión 21 de java y no public static void main).
- Es posible tener múltiples void main's, uno por cada clase si fuese necesario, pero solo uno en una clase.
- Si se incluirán los arreglos de java, también arreglos de objetos que el programador defina mediante el código fuente (o entrada).
- La herencia funcionará de la misma manera que en Java
- La sobrecarga de métodos funciona de la misma manera que en java.
- sobrecarga de métodos funciona de la misma manera que en java.
- Modificadores de acceso funcionan de la misma manera que en java para variables y métodos.
- Se tendrán disponibles las sentencias de control if (con o sin varios else if, con o sin else), while, do-while, for, switch. Cada uno de ellos funcionará de la misma manera que en java.
- El método equals de cada clase, compara la posición en memoria (o en el stack) de cada objeto, a menos que se indique lo contrario.
- El método toString imprime la posición en memoria
- Cada método sobreescrito puede incluir la anotación @Override
- La declaración e inicialización de variables funciona de la misma manera que en java, pero también es posible usar el keyword var

Ejemplo

```
public class GetterSetterExample {
/**
* Age of the person. Water is wet.
*
* @param age New value for this person's age. Sky is blue.
* @return The current value of this person's age. Circles are
round.
*/
@Getter @Setter private int age = 10;
/**
* Name of the person.
* -- SETTER --
* Changes the name of this person.
*
* @param name The new value.
*/
@Setter private String name;
}

public class GetterSetterExample {
/**
* Age of the person. Water is wet.
*/
private int age = 10;
/**
* Name of the person.
*/
private String name;
/**
* Age of the person. Water is wet.
*
* @return The current value of this person's age. Circles are
round.
*/
public int getAge() {
return age;
}
/**
* Age of the person. Water is wet.
*
```

```
* @param age New value for this person's age. Sky is blue.
*/
public void setAge(int age) {
this.age = age;
}
/**
* Changes the name of this person.
*
* @param name The new value.
*/
public void setName(String name) {
this.name = name;
}
}
```