**MANUAL DE PROGRAMADOR**

**UNIVERSIDAD NACIONAL JOSE MARIA ARGUEDAS ALTAMIRANO**

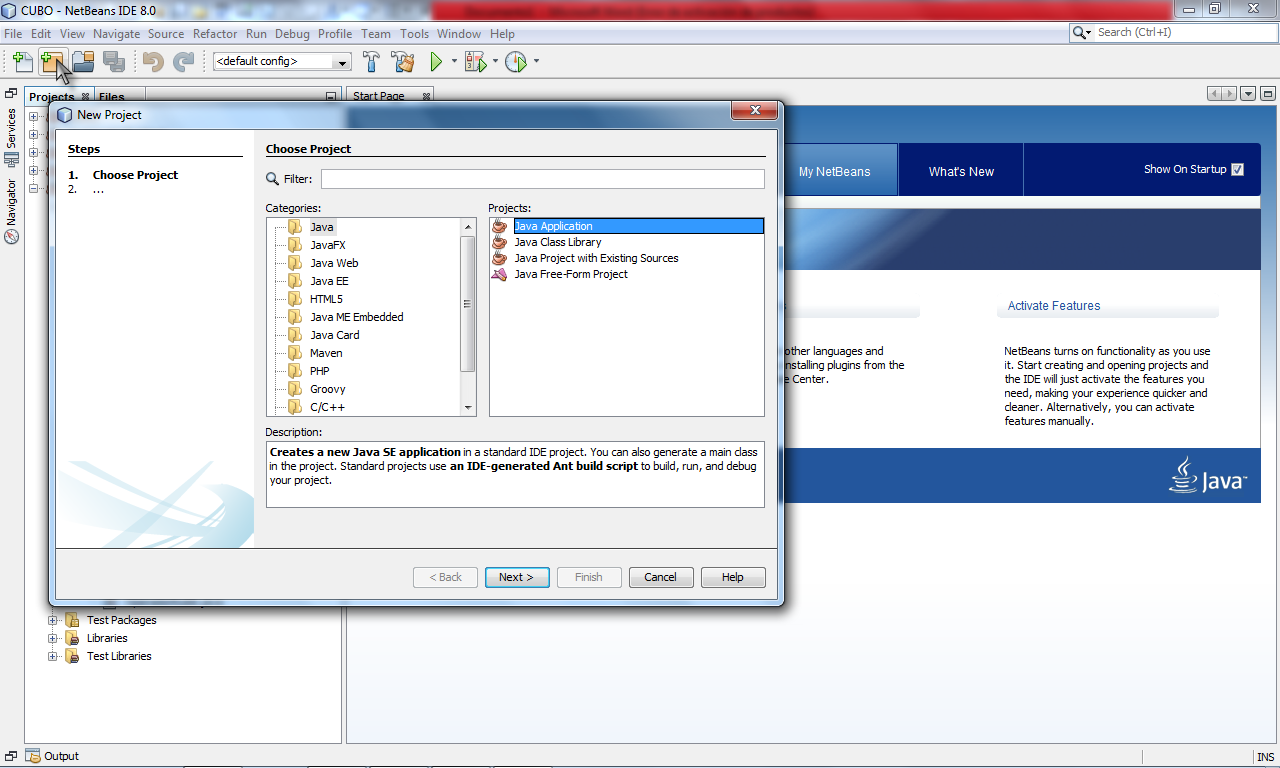
**INGENIERIA DE SISTEMAS**

**SOFTWARE III**

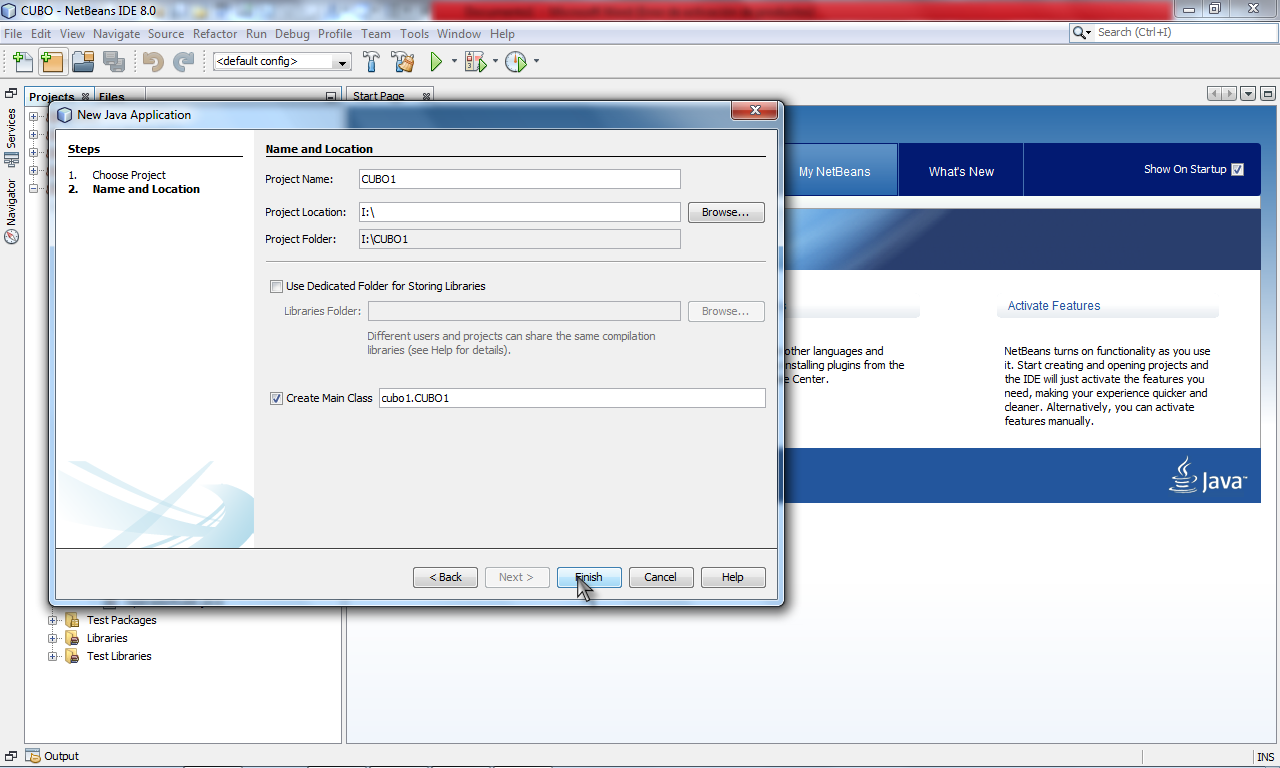
* **ROMULO QUISPE ALTAMIRANO.**
* **ALEX JAVIER HUAYLLAS CHIRCCA.**
* **YAISON HUAMAN MALLQUI.**

El proyecto CUBO esta esta programado con el lenguaje de programcion java, la codificasion se iso en el entorno de trabajo netBeans IDE 8.0

Una ves abierto el netBeans IDE 8.0, creamos una nueva categoria java y un proyecto de java aplicasion asi como se muestra en la imagen.



El nombre del proyecto llamaremos CUBO y también podemos direccionarlo en donde tú quieras que se guarde el proyecto o si no solamente pones finish y se guardara por defecto en el disco C.

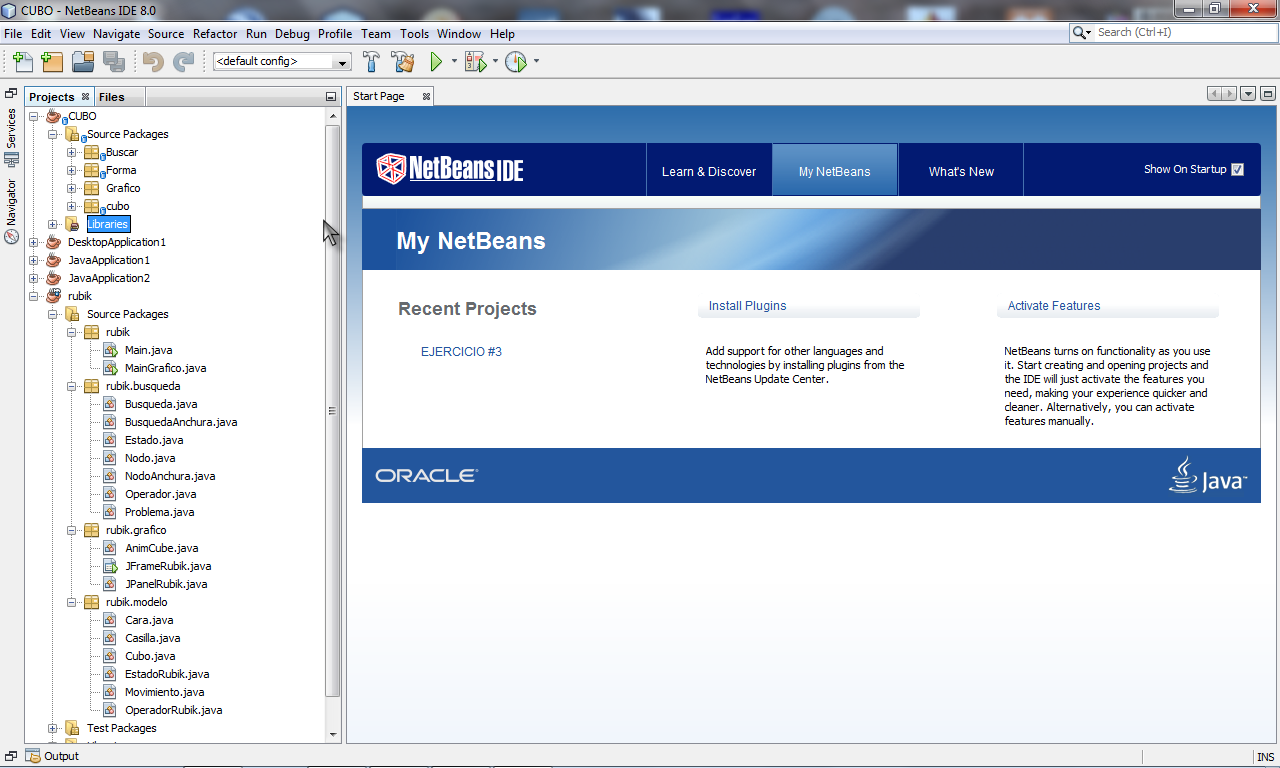


Una ves creado el proyecto CUBO creamos las siguentes paquetes para que nuestra aplicasion CUBO sea mas ordenado.

**Paquete Buscar.-**Clases abstractas e interfaces genéricos utilizados para la implementación de algoritmos de búsqueda en espacios de estados.

**Paquete Forma.-** Clases para modelizar y gestionar un Cubo.

**Paquete Grafico.-**Interfaz gráfico basado en el applet.



**Clases:**

**Problema:** Problema genérico a resolver mediante búsqueda en espacio de estados.

Caracterizado por un Estado inicial y el algoritmo de búsqueda a emplear

**Nodo:** Superclase con la definición de los nodos (contienen Estados) a usar en los algoritmos de búsqueda

Para manejar la información extra utilizada por un algoritmo de búsqueda concreto se deberá definir una subclase específica de la case Nodo.

**Interfaces**

**Busca:** Interfaz a implementar por los métodos de búsqueda

**Estado:** Interfaz a implementar por los objetos que almacenen estados del problema

**Operar:** Interfaz a implementar por los objetos que tomen el papel de operadores

Se incluye también una implementación de la Búsqueda en anchura como ejemplo del uso de estas clases e interfaces.

**Clases.**

**Busca Anchura**: Implementación de la búsqueda en anchura haciendo uso de listas ABIERTOS y CERRADOS

**Nodo Anchura:** Definición de los nodos a usar en las búsquedas en anchura.

En cada nodo se toma nota del operador empleado para generar el estado asociado (en el nodo raíz es null), para facilitar la generación de la lista de operadores resultante de una búsqueda satisfactoria.

**Código de la clase Problema.**

package Buscar;

import java.util.Vector;

public class Problema {

private Estado inicio;

private Busca busca;

public Problema(Estado inicia, Busca busca) {

this.inicio = inicia;

this.busca = busca;

}

public void setBuscador(Busca busca) {

this.busca = busca;

}

public void setInicial(Estado inicia) {

this.inicio = inicia;

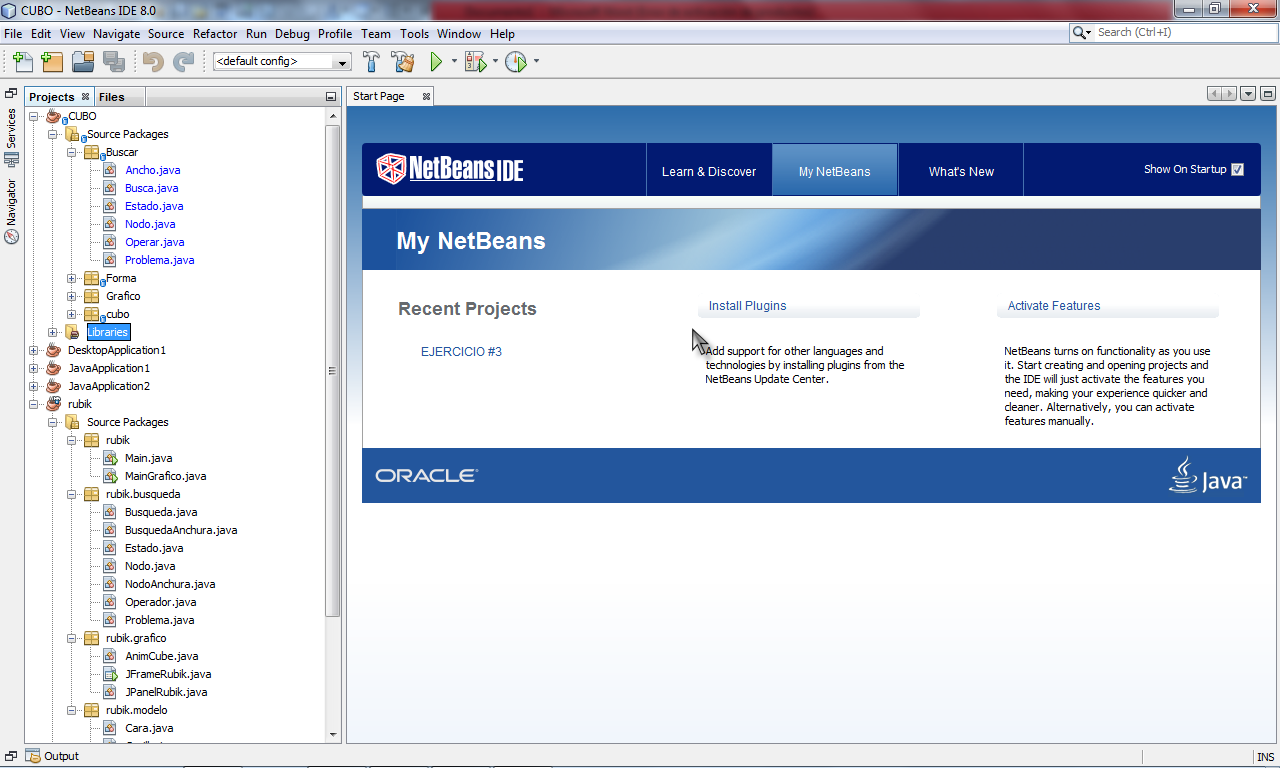
}

public Vector<Operar> obtenerSolucion() {

return(busca.buscarSolucion(inicio));

}

}



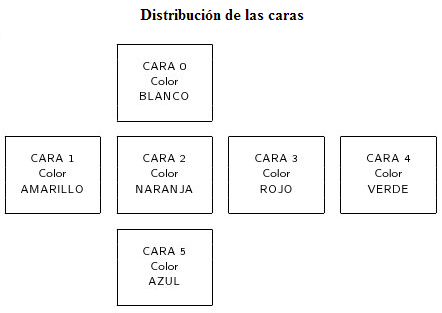
**Clases:**

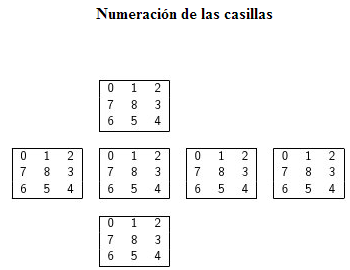
**Cubo:** Encapsula la información sobre un cubo e información adicional para gestionarlo.

* Cada Cubo está formado por 6 Caras
* Incorpora métodos para duplicar y comparar Cubos, determinar si es solución y aplicar movimientos sobre las caras.
* Incluye arrays estático con información sobre la vecindad de cada cara y los índices de las casillas ''fronterizas'' de cada cara vecina

**Cara:** Encapsula la información sobre una cara de un cubo e información adicional para gestionarla.

* Cada Cara almacena en un array de 9 posiciones la situación de sus casillas
* La distribución de las Casillas en ese array sigue la distribución *''en espiral''* que se muestra en las gráficas siguientes





**Casilla:** Encapsula la información sobre una casilla de una cara de un cubo.

Para cada casilla se almacenan dos atributos

**Color:** un valor de tipo byte que identifica el color de esa casilla. Es decir, nos indica implíctamente en que cara debería de estar situada.

**Posición Correcta:** un valor de tipo byte que indica la posición que esa casilla debería ocupar en el array de casillas asociado a la cara a la que pertenece, conforme a la la distribución *''en espiral''*.

**Movimiento:** Encapsula los movimientos permitidos sobre los cubos.

Por razones de eficiencia los objetos correspondientes a los 12 movimientos posibles ya estarán creados y se almacenan en atributos estáticos de esta clase

Se incluyen también las clases necesarias para integrar este modelo de Cubo 3x3x3 en el *framework* de búsqueda.

**Clases:**

**Estado:** Implementación del interfaz busca.Estado, contiene un objeto Forma.Cubo en el que delega las operaciones del interfaz Estado

Por razones de eficiencia, un array con los 12 operadores aplicables se almacenan en un vector estático asociado a la clase para no tener que crearlo cada vez que se le pida al Estado la lista de operadores que son aplicables sobre él.

**Operador:** Implementación del interfaz busca.Operar, contiene un objeto Forma.Movimiento

**Código de la clase Casilla**

package Forma;

public class Casilla {

public byte color;

public byte posicionCorrecta;

public Casilla(byte color, byte posicionCorrecta) {

this.color = color;

this.posicionCorrecta = posicionCorrecta;

}

public byte getColor() {

return color;

}

public void setColor(byte color) {

this.color = color;

}

public byte getPosicionCorrecta() {

return posicionCorrecta;

}

public void setPosicionCorrecta(byte posicionCorrecta) {

this.posicionCorrecta = posicionCorrecta;

}

@Override

public boolean equals(Object o) {

if (o == null) {

return false;

}

if (getClass() != o.getClass()) {

return false;

}

final Casilla other = (Casilla) o;

if ((this.color != other.color) ||

(this.posicionCorrecta != other.posicionCorrecta)) {

return false;

}

return true;

}

@Override

public int hashCode() {

int hash = 7;

hash = 10 \* hash + this.color;

hash = 17 \* hash + this.posicionCorrecta;

return (10 \* this.color + this.posicionCorrecta);

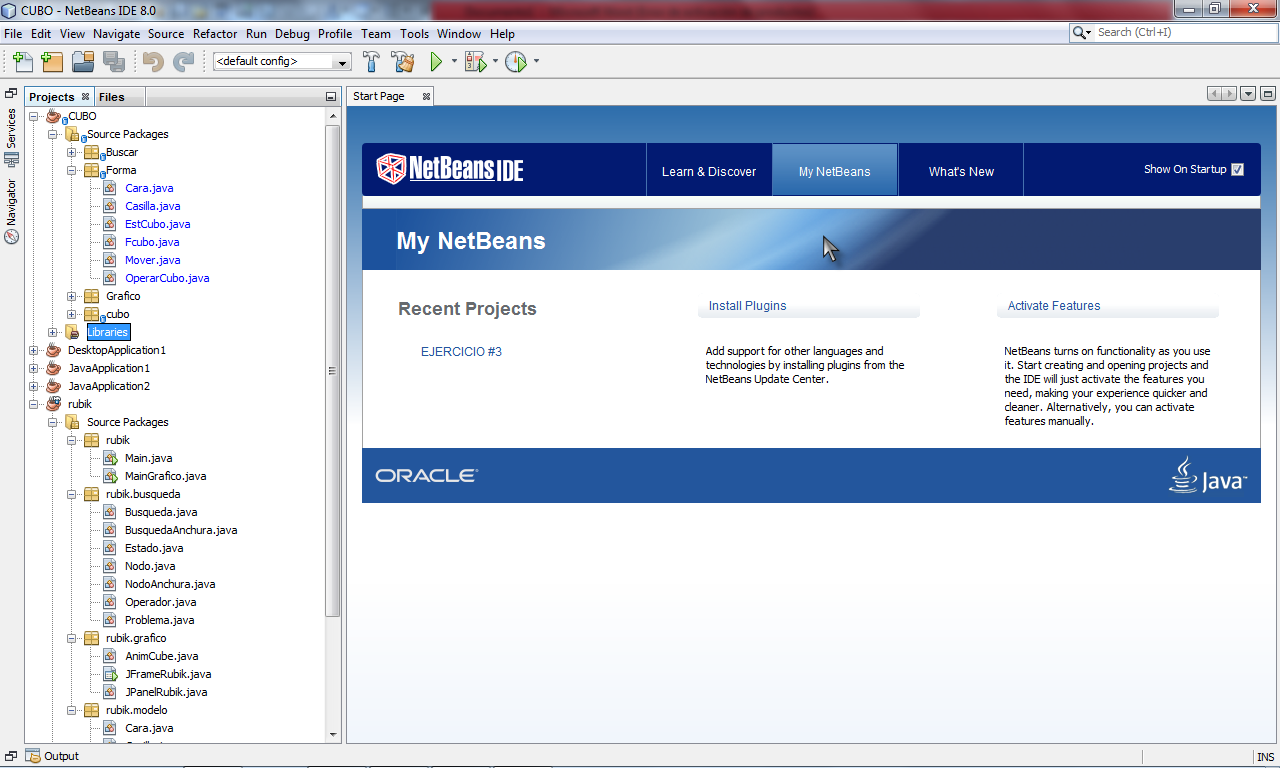
}

public Casilla clone() {

return (new Casilla(this.color, this.posicionCorrecta));

}

}



**Clases:**

**JPanelRubik:** Encapsula al applet AnimCube en un JPanel, añadiendo métodos para recuperar el cubo mostrado en pantalla y para establecer su configuración

**JFrameRubik:** Ejemplo de ventana (JFrame) donde se usa JPanelCubo para mostrar y manipular gráficamente un Cubo.

**Código de la clase jpaCubo**

package Grafico;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.util.Vector;

import javax.swing.JPanel;

import Forma.Fcubo;

import Forma.Mover;

public class JpCubo extends JPanel {

private AnimCube animCubo;

private Fcubo cubo;

public JpCubo() {

this(new Fcubo());

}

public JpCubo(Fcubo c) {

super();

cubo = c;

animCubo = new AnimCube();

animCubo.setPreferredSize(new Dimension(250, 250));

animCubo.init();

animCubo.setCubo(c);

this.setLayout(new BorderLayout());

this.add(animCubo, BorderLayout.CENTER);

JPanel aux = new JPanel();

}

public Fcubo getCubo() {

return (animCubo.getCubo());

}

public void setCubo(Fcubo cubo) {

this.animCubo.setCubo(cubo);

}

public void animarCubo(Vector<Mover> movimientos) {

Fcubo cuboActual = animCubo.getCubo();

for (Mover m : movimientos){

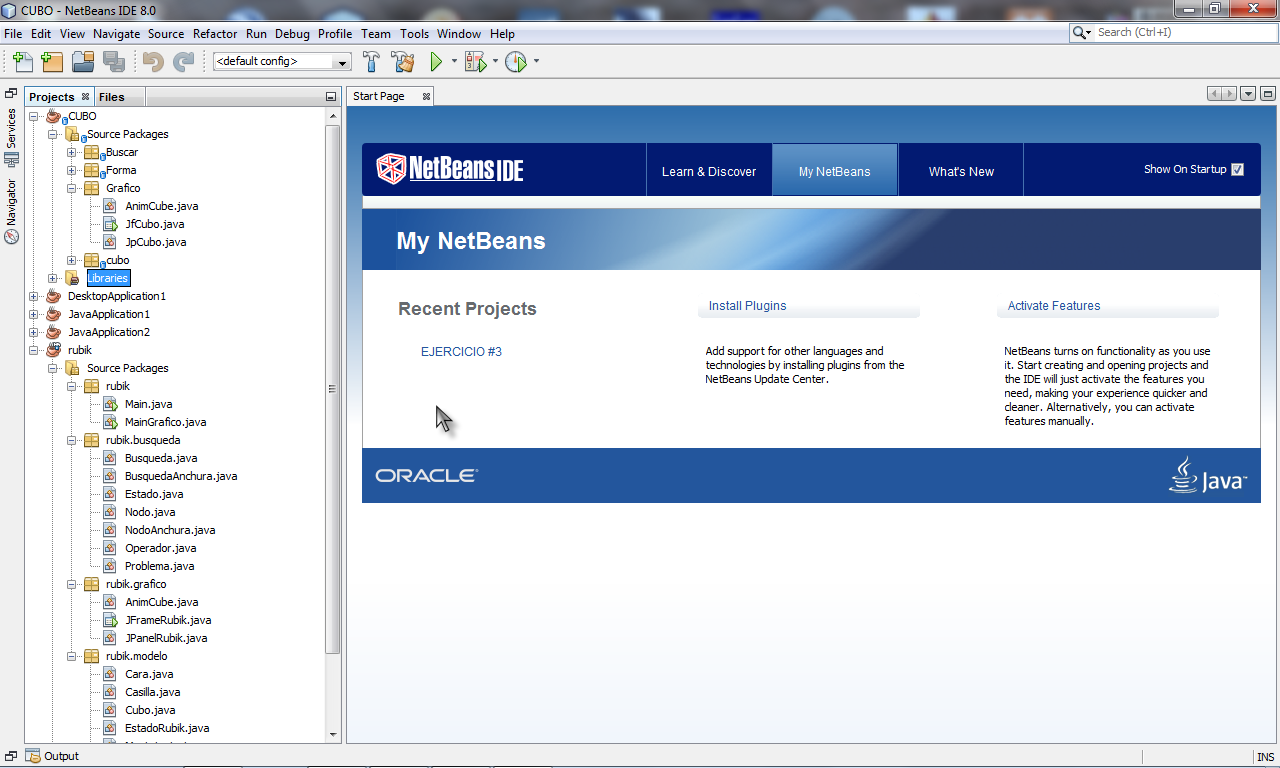
cuboActual.mover(m);

animCubo.setCubo(cuboActual);

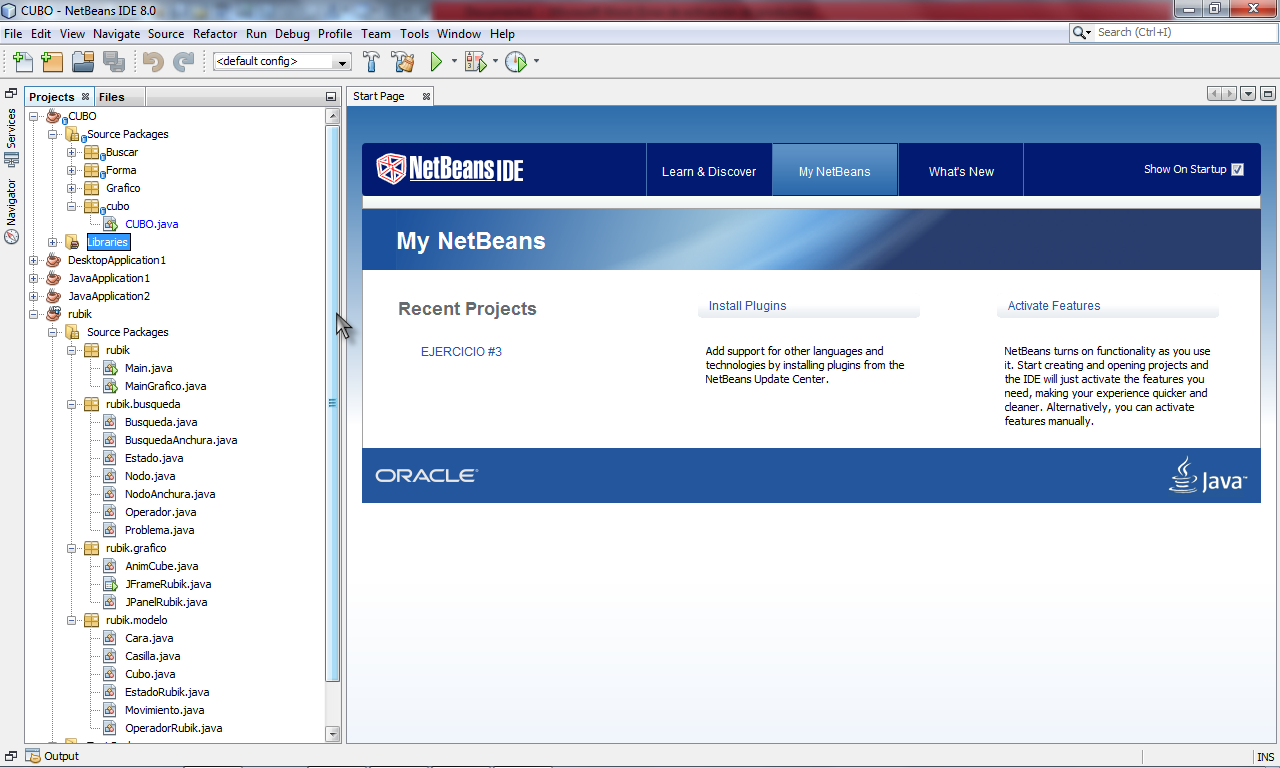
}

}

}



En el paquete CUBO se encuentra la clase principal que la clase Main, aquí es lo que se ejecuta el CUBO.



Por ultimo para que nuestro CUBO funcione se agrega dos librerías como se muestra en la imagen.

