



INSTITUTO POLITECNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO (ESCOM)



PROGRAMACIÓN ORIENTADA A OBJETOS

NOMBRE DEL ALUMNO:

- SANTOS MÉNDEZ ULISES JESÚS

PRÁCTICA:

- JAVA 3D

NÚMERO DE PRÁCTICA: 3

OPCIÓN 1:

- SISTEMA SOLAR(PLANETARIO)

FECHA DE ENTREGA:

- 27/05/2021

GRUPO:

- 2CM11

Java 3D

Introducción

El API Java 3D es una interface para escribir programas que muestran e interactúan con gráficos tridimensionales. El API Java 3D proporciona una colección de constructores de alto nivel para crear y manipular geometrías 3D y estructuras para dibujar esa geometría, de la misma manera proporciona las funciones para creación de imágenes, visualizaciones, animaciones y programas de aplicaciones gráficas 3D interactivas.

API 3D de JAVA

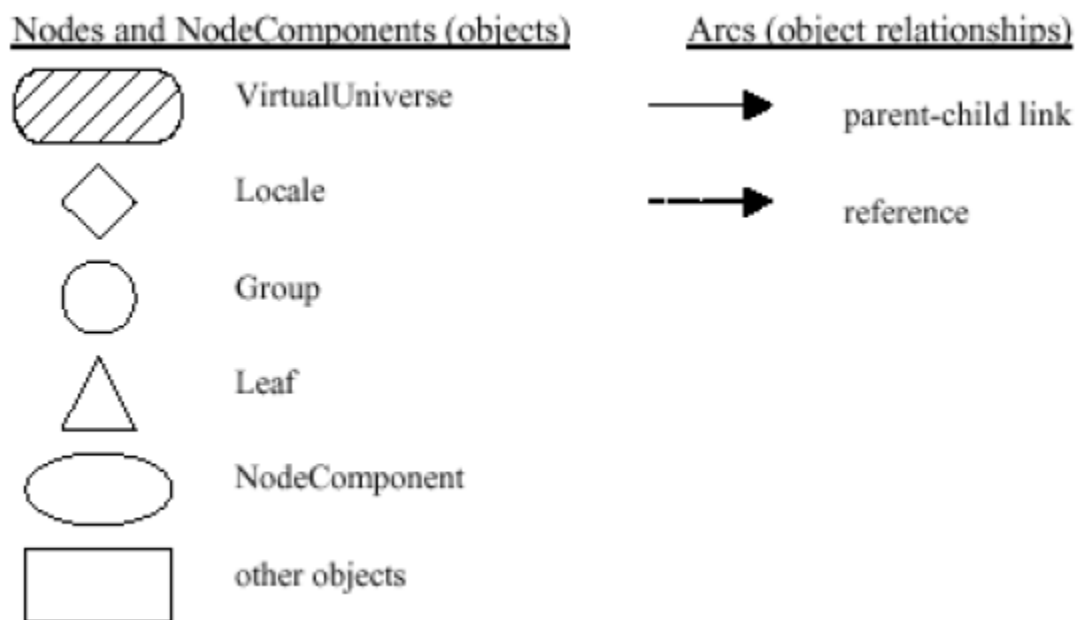
Es un árbol de clases Java que sirven como interface para sistemas de renderizado de gráficos tridimensionales y un sistema de sonido. El programador trabaja con constructores de alto nivel para crear y manipular objetos geométricos en 3D. Estos objetos geométricos residen en un universo virtual, que luego es renderizado. El API está diseñado con flexibilidad para crear universos virtuales precisos de una amplia variedad de tamaños.

Universo Virtual

Un universo virtual Java 3D se crea desde un escenario gráfico. Un escenario gráfico se crea usando ejemplares de clase Java 3D, el escenario gráfico está ensamblado desde objetos que definen la geometría, sonidos, orientación, etc.

Las representaciones gráficas de un escenario gráfico pueden servir como herramienta de diseño y/o documentación para los programas Java 3D.

Para diseñar un universo virtual Java 3D se dibuja un escenario gráfico usando un conjunto de símbolos estándar. Después de completar el diseño, este escenario gráfico es la especificación del programa.



Desarrollo

1. Se importaron las siguientes librerías de java para el diseño del universo en Java 3D, en ella están contenidas las clases y métodos que se utilizarán.

```
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import javax.media.j3d.*;
import java.util.*;
import javax.swing.event.*;
import javax.swing.JFrame;
import java.awt.*;
import java.awt.event.*;
import com.sun.j3d.utils.image.TextureLoader;
import javax.swing.WindowConstants;
import javax.vecmath.*;
```

2. Se declaró la clase publica SolarSis

```
public class solarsis {
    public solarsis(){
```

3. Se crean grupos con **BranchGroup** que es la raíz de un subgráfico o rama gráfica, la rama de contenido gráfico especifica el contenido del universo virtual, geometría, apariencia, etc.

```
BranchGroup group = new BranchGroup();
Appearance appsol = new Appearance();
Appearance appmercurio = new Appearance();
Appearance appvenus = new Appearance();
Appearance apptierra = new Appearance();
Appearance appluna = new Appearance();
Appearance appmarte = new Appearance();
Appearance appjupiter = new Appearance();
Appearance appsaturno = new Appearance();
Appearance appurano = new Appearance();
Appearance appneptuno = new Appearance();
Appearance apppluton = new Appearance();
```

4. Se utiliza el objeto **TextureLoader** para el uso de ficheros como formatos, se pueden cargar JPEG, GIF, PNG entre otros.

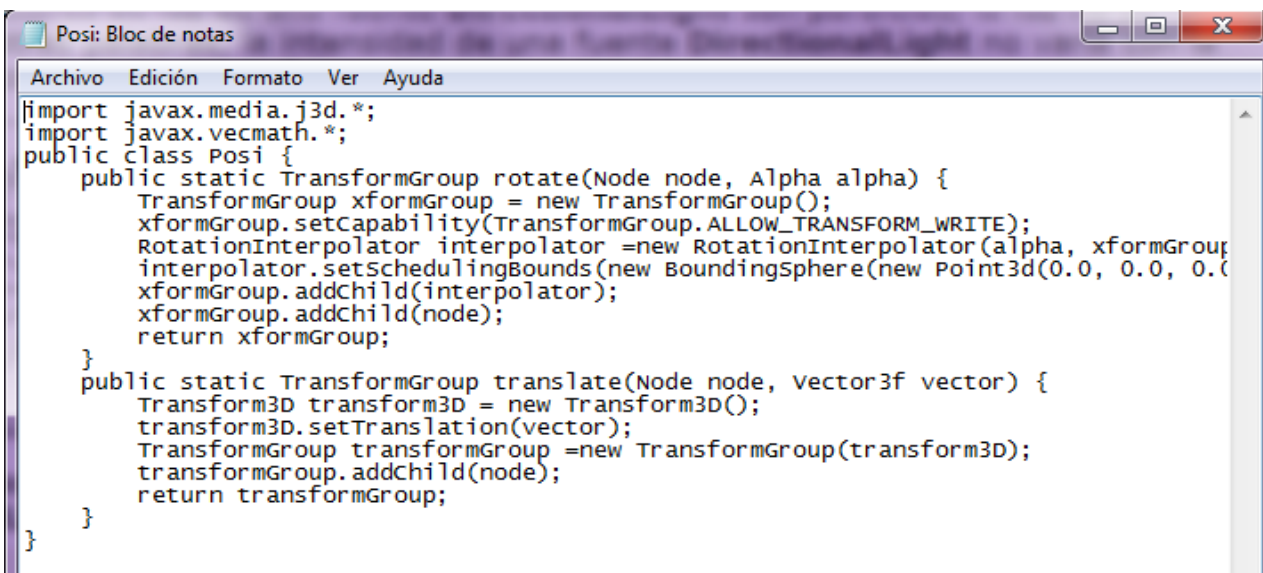
```
TextureLoader tex = new TextureLoader("sol.jpg", null);
appsol.setTexture(tex.getTexture());
tex = new TextureLoader("Mercurio.jpg", null);
appmercurio.setTexture(tex.getTexture());
tex = new TextureLoader("Venus.jpg", null);
appvenus.setTexture(tex.getTexture());
tex = new TextureLoader("Tierra.jpg", null);
apptierra.setTexture(tex.getTexture());
tex = new TextureLoader("Luna.jpg", null);
appluna.setTexture(tex.getTexture());
tex = new TextureLoader("Marte.jpg", null);
appmarte.setTexture(tex.getTexture());
tex = new TextureLoader("Jupiter.jpg", null);
appjupiter.setTexture(tex.getTexture());
tex = new TextureLoader("Saturno.jpg", null);
appsaturno.setTexture(tex.getTexture());
tex = new TextureLoader("Urano.jpg", null);
appurano.setTexture(tex.getTexture());
tex = new TextureLoader("Neptuno.jpg", null);
appneptuno.setTexture(tex.getTexture());
tex = new TextureLoader("Pluton.jpg", null);
appluton.setTexture(tex.getTexture());
```

5. Se utilizan clases de utilidad geométrica para crear gráficos primitivos como cajas, conos, cilindros y esferas. La clase Sphere crea objetos visuales esféricos con el centro en el origen.

```
Sphere sol = new Sphere(0.696f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, appsol);
Sphere mercurio = new Sphere(0.0243f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, appmercurio);
Sphere venus = new Sphere(0.0605f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, appvenus);
Sphere tierra = new Sphere(0.0637f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, apptierra);
Sphere luna = new Sphere(0.0173f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, appluna);
Sphere marte = new Sphere(0.0338f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, appmarte);
Sphere jupiter = new Sphere(0.0699f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, appjupiter);
Sphere saturno = new Sphere(0.0582f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, appsaturno);
Sphere urano = new Sphere(0.0253f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, appurano);
Sphere neptuno = new Sphere(0.0246f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, appneptuno);
Sphere pluton = new Sphere(0.0118f, Primitive.GENERATE_NORMALS |
Primitive.GENERATE_TEXTURE_COORDS, 32, appluton);
```

6. Se hacen girar los planetas sobre su propio eje, usando **Alpha** puede posicionar, orientar, y escalar cada uno de los nodos hijos, y se agrega duración y tiempo, se hace uso del método rotate en Posi.java, se hace uso de **RotationInterpolator** para que se pueda variar la orientación rotacional de un objeto visual sobre un eje.

```
TransformGroup solRotXformGroup =
Posi.rotate(sol, new Alpha(-1, 1250));
TransformGroup mercurioRotXformGroup =
Posi.rotate(mercurio, new Alpha(-1, 1350));
TransformGroup venusRotXformGroup =
Posi.rotate(venus, new Alpha(-1, 1800));
TransformGroup tierraRotXformGroup =
Posi.rotate(tierra, new Alpha(-1, 1300));
TransformGroup lunaRotXformGroup =
Posi.rotate(luna, new Alpha(-1, 1300));
TransformGroup marteRotXformGroup =
Posi.rotate(marte, new Alpha(-1, 1250));
TransformGroup jupiterRotXformGroup =
Posi.rotate(jupiter, new Alpha(-1, 1250));
TransformGroup saturnoRotXformGroup =
Posi.rotate(saturno, new Alpha(-1, 1200));
TransformGroup uranoRotXformGroup =
Posi.rotate(urano, new Alpha(-1, 1200));
TransformGroup neptunoRotXformGroup =
Posi.rotate(neptuno, new Alpha(-1, 1350));
TransformGroup plutonRotXformGroup =
Posi.rotate(pluton, new Alpha(-1, 1200));
```



```
Posi: Bloc de notas
Archivo Edición Formato Ver Ayuda
import javax.media.j3d.*;
import javax.vecmath.*;
public class Posi {
    public static TransformGroup rotate(Node node, Alpha alpha) {
        TransformGroup xformGroup = new TransformGroup();
        xformGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        RotationInterpolator interpolator = new RotationInterpolator(alpha, xformGroup);
        interpolator.setSchedulingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0));
        xformGroup.addChild(interpolator);
        xformGroup.addChild(node);
        return xformGroup;
    }
    public static TransformGroup translate(Node node, vector3f vector) {
        Transform3D transform3D = new Transform3D();
        transform3D.setTranslation(vector);
        TransformGroup transformGroup = new TransformGroup(transform3D);
        transformGroup.addChild(node);
        return transformGroup;
    }
}
```

7. Se hace uso del método `translate` en `Posi.java`, se hace uso de la clase **Transform3D** y **setTranslation** y reemplaza los componentes de translación con los valores en el vector3D

```
TransformGroup mercurioTransXformGroup =
Posi.translate(mercurioRotXformGroup, new Vector3f(0.0f, 0.0f, 0.3f));
TransformGroup venusTransXformGroup =
Posi.translate(venusRotXformGroup, new Vector3f(0.0f, 0.0f, 0.5f));
TransformGroup tierraTransXformGroup =
Posi.translate(tierraRotXformGroup, new Vector3f(0.0f, 0.0f, 0.6f));
TransformGroup lunaTransXformGroup =
Posi.translate(lunaRotXformGroup, new Vector3f(0.0f, 0.0f, 0.6f));
TransformGroup marteTransXformGroup =
Posi.translate(marteRotXformGroup, new Vector3f(0.0f, 0.0f, 0.7f));
TransformGroup jupiterTransXformGroup =
Posi.translate(jupiterRotXformGroup, new Vector3f(0.0f, 0.0f, 0.95f));
TransformGroup saturnoTransXformGroup =
Posi.translate(saturnoRotXformGroup, new Vector3f(0.0f, 0.0f, 1.1f));
TransformGroup uranoTransXformGroup =
Posi.translate(uranoRotXformGroup, new Vector3f(0.0f, 0.0f, 1.5f));
TransformGroup neptunoTransXformGroup =
Posi.translate(neptunoRotXformGroup, new Vector3f(0.0f, 0.0f, 1.9f));
TransformGroup plutonTransXformGroup =
Posi.translate(plutonRotXformGroup, new Vector3f(0.0f, 0.0f, 1.95f));
```

8. Se utiliza la clase `TransformGroup` para hacer rotar los planetas alrededor del Sol

```
TransformGroup mercurioRotGroupXformGroup =
Posi.rotate(mercurioTransXformGroup, new Alpha(-1, 3000));
TransformGroup venusRotGroupXformGroup =
Posi.rotate(venusTransXformGroup, new Alpha(-1, 5000));
TransformGroup tierraRotGroupXformGroup =
Posi.rotate(tierraTransXformGroup, new Alpha(-1, 6500));
TransformGroup lunaRotGroupXformGroup =
Posi.rotate(lunaTransXformGroup, new Alpha(-1, 6500));
TransformGroup marteRotGroupXformGroup =
Posi.rotate(marteTransXformGroup, new Alpha(-1, 7300));
TransformGroup jupiterRotGroupXformGroup =
Posi.rotate(jupiterTransXformGroup, new Alpha(-1, 8300));
TransformGroup saturnoRotGroupXformGroup =
Posi.rotate(saturnoTransXformGroup, new Alpha(-1, 8500));
TransformGroup uranoRotGroupXformGroup =
Posi.rotate(uranoTransXformGroup, new Alpha(-1, 8960));
TransformGroup neptunoRotGroupXformGroup =
Posi.rotate(neptunoTransXformGroup, new Alpha(-1, 9000));
TransformGroup plutonRotGroupXformGroup =
Posi.rotate(plutonTransXformGroup, new Alpha(-1, 9500));
```

9. Se usa la clase `Group` para agrupar nodos teniendo un padre y los correspondientes hijos y se utiliza `addChild` para agregar los nodos a la lista de nodos.

```
group.addChild(solRotXformGroup);
group.addChild(mercurioRotGroupXformGroup);
group.addChild(venusRotGroupXformGroup);
group.addChild(tierraRotGroupXformGroup);
group.addChild(lunaRotGroupXformGroup);
group.addChild(marteRotGroupXformGroup);
group.addChild(jupiterRotGroupXformGroup);
group.addChild(saturnoRotGroupXformGroup);
group.addChild(uranoRotGroupXformGroup);
group.addChild(neptunoRotGroupXformGroup);
group.addChild(plutonRotGroupXformGroup);
```

10. Se hace uso del objeto SimpleUniverse para crear una rama de vista gráfica completa para un universo virtual. En la rama se incluye un plato de imagen en donde se proyectará la imagen renderizada, y el objeto **Canvas3D** proporciona una imagen en la ventana de nuestra pantalla.

```
GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
Canvas3D canvas = new Canvas3D(config);
canvas.setSize(1000,700);
SimpleUniverse universe = new SimpleUniverse(canvas);
universe.getViewingPlatform().setNominalViewingTransform();
universe.addBranchGraph(group);
```

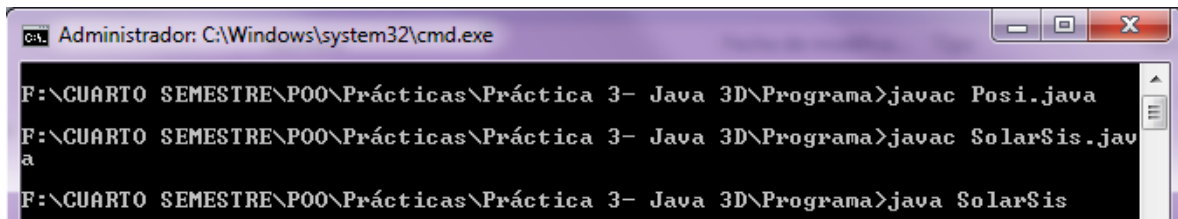
11. Se crea la ventana donde se agregará a canvas y se hará visible

```
JFrame f = new JFrame("Planetario");
f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
f.add(canvas);
f.pack();
f.setVisible(true);
```

12. Por último se agrega el método main

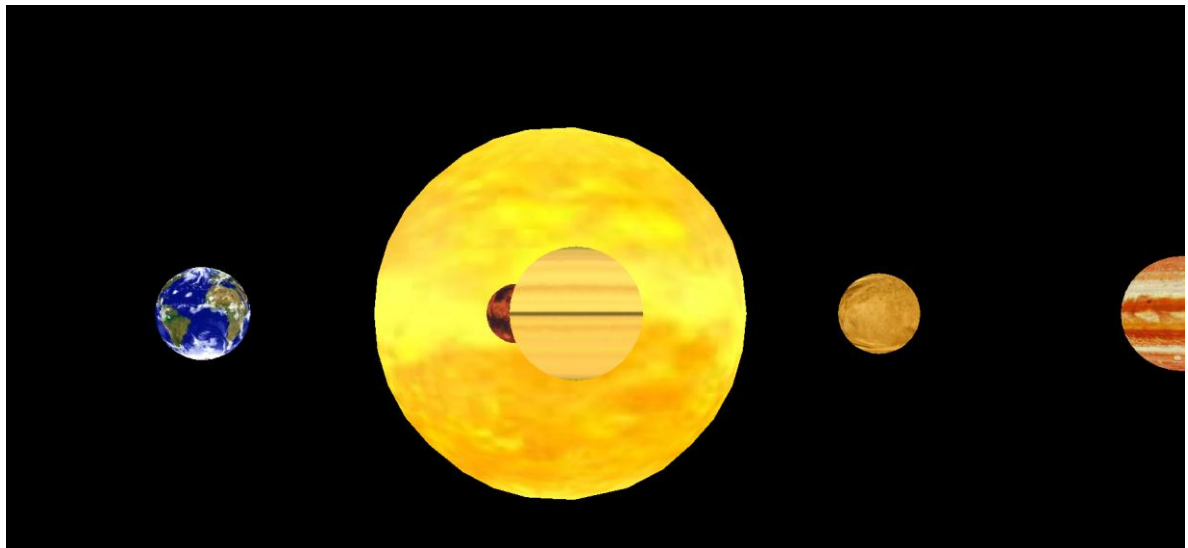
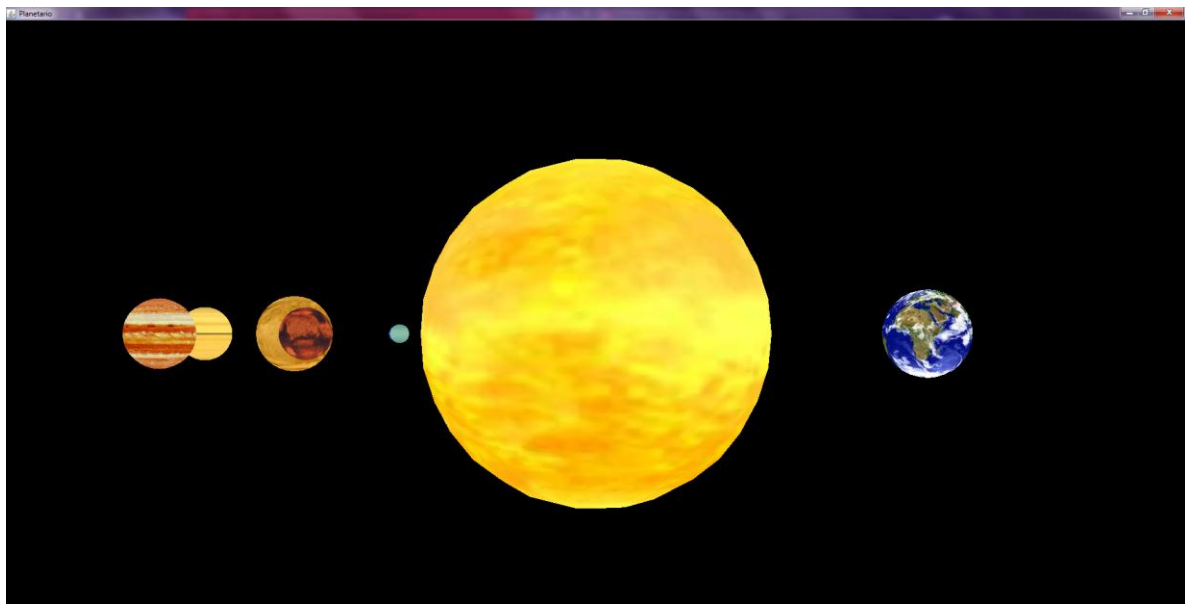
```
}
public static void main(String[] args){
    SolarSis solar = new SolarSis();
}
}
```

13. Se procede a compilar y a ejecutar el programa en la consola CMD, aclarando que al inicio se tuvo un error pero se logró corregir copiando la carpeta lib de Java3D y pegándola en el jre de la versión de Java, de esta manera se eliminaron los mensajes de error del programa.



```
Administrador: C:\Windows\system32\cmd.exe

F:\CUARTO SEMESTRE\POO\Prácticas\Práctica 3- Java 3D\Programa>javac Posi.java
F:\CUARTO SEMESTRE\POO\Prácticas\Práctica 3- Java 3D\Programa>javac SolarSis.java
F:\CUARTO SEMESTRE\POO\Prácticas\Práctica 3- Java 3D\Programa>java SolarSis
```



Conclusión

En conclusión esta práctica fue útil para conocer el proceso de la creación de objetos 3D a partir de una parte teórica y de ejemplos donde pudimos entender el funcionamiento de los objetos, clases y métodos empleados para renderizar, modelar, y darle consistencia a nuestra figura utilizando librerías de java3D.