23 | Sistema solar

Prerrequisitos

- Herencia

Meta

Que el alumno desarrolle una aplicación completa aplicando el diseño orientado a objetos para resolver un problema donde cooperan varias instancias de diversas clases.

Objetivos

Al finalizar la práctica el alumno será capaz de:

- Describir el diseño de un proyecto completo utilizando UML.
- Explicar cómo se distribuyen tareas entre diferentes instancias(objetos) de diversas clases.
- Crear una animación en `JavaFX`.

Antecedentes

`JavaFX` es un conjunto de paquetes gráficos y multimedia (video, audio, internet e interfaces gráficas de usuario) para diseñar, crear, probar, purgar y desplegar aplicaciones que operan en cualquier sistema operativo. Su diseño está inspirado en técnicas estándar para la produción de animaciones.

Actividad 23.1

JavaFX no es parte de la distribución de Java, así que es necesario instalarlo por separado. Sigue las instrucciones en https://openjfx.io/openjfx-docs/#install-javafx. Puedes utilizar la última versión disponible. Para compilar y ejecutar el código ya no funciona ant, por lo que se te proporciona un archivo Makefile para utilizar ahora la herramienta make. Este archivo manda llamar los comandos manuales para compilar y ejecutar el código. También puede resultarte útil conocerlos.

Ejecuta los comandos siguientes en la consola, observa que debes ajustarlos según el directorio donde descargaste JavaFX y la versión que tengas.

```
$ export PATH_TO_FX=<directorio_de_JavaFX>/javafx-sdk-11.0.1/lib
$ javac --module-path $PATH_TO_FX --add-modules=javafx.controls
    ↪ -d ./classes --source-path ./src src/sistemasolar/
    ↪ SistemaSolar.java
$ java --module-path $PATH_TO_FX --add-modules=javafx.controls -
    ↪ classpath classes sistemasolar.SistemaSolar
```

Utiliza make para compilar, ejecutar y limpiar el código generado:

```
$ make compile
$ make run
$ make clean
```

Arquitectura del proyecto

Para programar una aplicación de JavaFX se debe comenzar con tres elementos:

Application La clase principal debe extender javafx.application.Application. La clase base se encargará de toda la configuración previa al lanzamiento de la interfaz de usuario, incluyendo todas las llamadas al sistema operativo, que se esconden detrás. El código correspondiente a tu programa comenzará a ejecutarse a partir del método public void start(Stage primaryStage) throws Exception, que debes sobreescribir.

Stage El método start de la clase Application recibe como parámetro un *escenario*. Esta clase representa al lugar donde se montará tu *escena*. Una aplicación puede tener varias escenas y éstas se van representando en el escenario, puedes alternar entre escenas, en forma análoga a lo que sucede en un teatro o una película.

Scene La *escena* es el contenedor para todos los objetos que se mostrarán al usuario, con sus animaciones.

Nodos

En `JavaFX` todos los elementos que aparecen en la pantalla se agregan a una escena y son objetos de tipo `Node`, incluso la cámara que *filma* la escena, aunque no la veamos. La Figura 24.5 muestra los elementos más importantes de la jerarquía de clases.

La documentación oficial se encuentra en `https://openjfx.io/javadoc/11/`, la necesitarás para consultar los detalles sobre los objetos y métodos de este proyecto.

Ejercicios

En este proyecto, sólo habrá una escena: el sistema solar con 9 planetas, los satélites más importantes y algunos asteroides (opcionalmente).

1. Para este proyecto se te entrega un aplicación de `JavaFX` que funciona. Ejecútala con `make compile; make run`.

   Tu trabajo será completar el sistema solar, para ello debes leer todo el código que te fue proporcionado y entender cómo funciona. La Figura 23.2 muestra el diagrama de clases, para que veas cómo está organizado el código.

2. Quita al satélite extra, la Luna no lo tiene.

3. Agrega a los planetas faltantes desde Mercurio hasta Neptuno. Los planetas actuales están a escala del sistema real aproximadamente, pero puede suceder que al agregar más planetas ya no quepan todos y no se vea bien. Es válido modificar los valores en aras de mejorar la estética, aunque la animación no sea astronómicamente correcta.

4. Ponle colores a los planetas, según sus colores reales. Se dará un punto extra a quien le ponga texturas con fotos de los planetas.

5. Agrega algunos satélites importantes, como los cuatro de Júpiter que vió Galileo con su telescopio.

6. Guarda una copia de tu sistema ya funcionando antes de proceder al siguiente paso, en caso de que algo salga mal.

7. Agrega a Plutón. El aspecto interesante de este planeta enano es que su órbita está inclinada con respecto a las demás, así que necesitarás entender el código que se te proporcionó, para modificarlo y agregar una opción que permita rotar la órbita. Esta parte de la práctica es obligatoria.

   Para lograr la rotación debes modificar el código que calcula las coordenadas (x, y) del planeta para cada tiempo. Una matriz de rotación te permitirá encontrar el nuevo valor de las coordenadas:

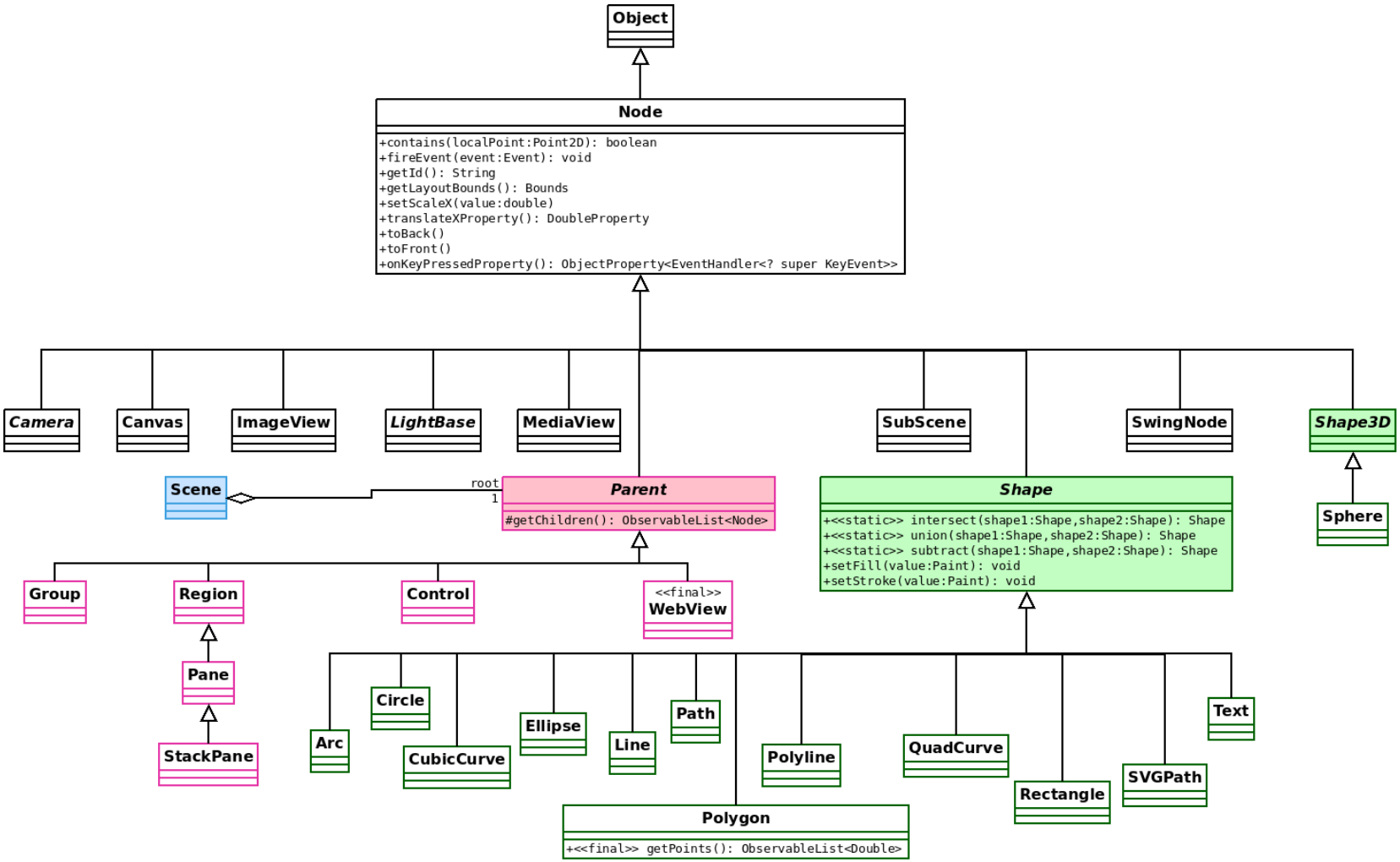

Figura 23.1 Diagrama de clases UML para los nodos de `JavaFX`.

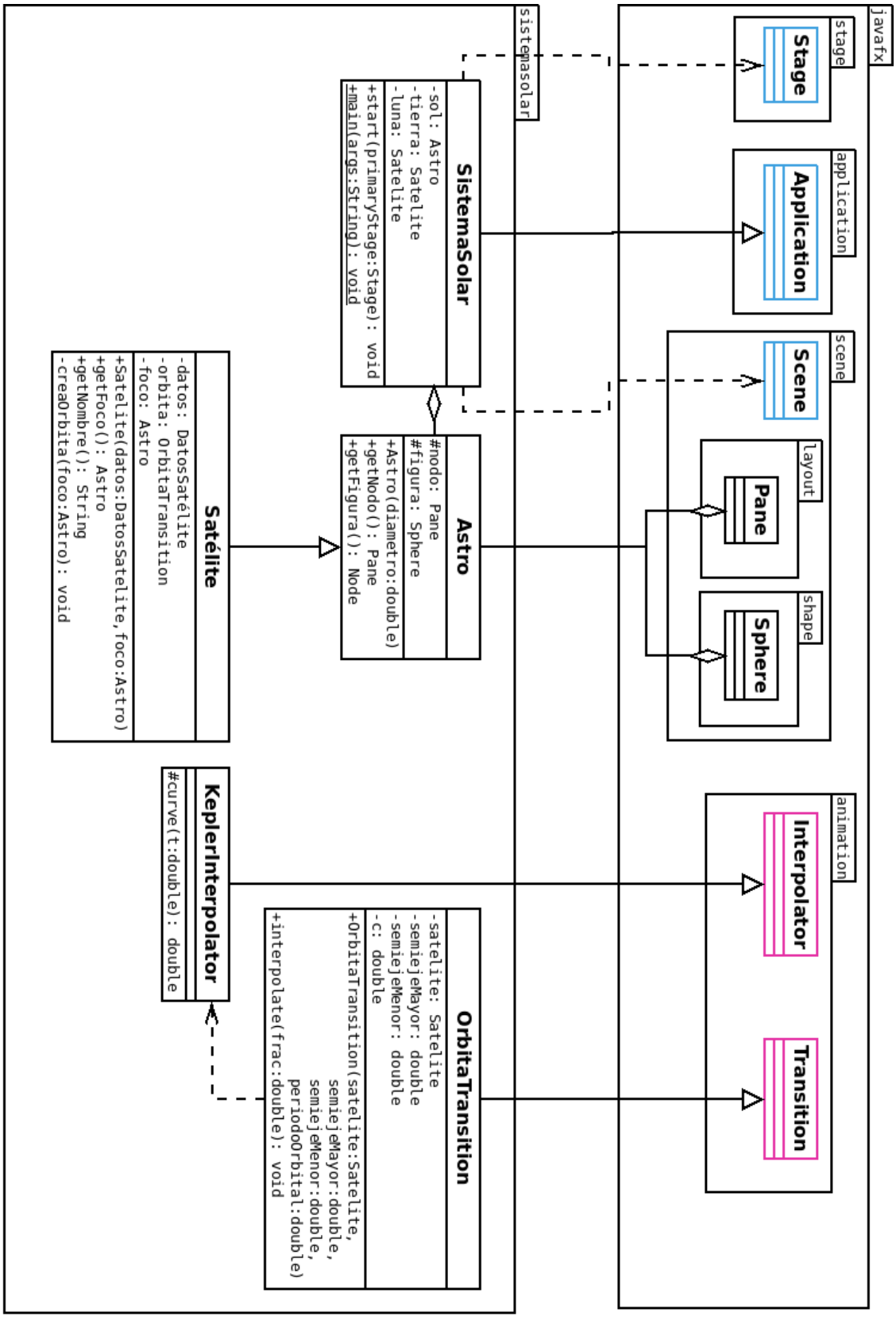

Figura 23.2 Arquitectura de la aplicación de `JavaFX` para modelar al sistema solar.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{23.1}$$

Observa que si $\alpha = 0$ entonces los valores de x y y no cambian.

8. Se dará otro punto extra a quien genere un número interesante de asteroides entre Marte y Júpiter. Deberás usar ciclos para ello y tal vez algún generador de números aleatorios para que los asteroides se vean un poco dispersos sobre la órbita.