FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Java

AlgoBlocks - JavaFX

Algoritmos y Programación III - [7507/9502]

Grupo 13 - Curso 2

Segundo cuatrimestre de 2020

Alumno	**Mail**
Apaldetti, Tomás	tapaldetti@fi.uba.ar
Kelman Axel	akelman@fi.uba.ar
Perticaro Agustin	aperticaro@fi.uba.ar
Gazzola Franco	fgazzola@fi.uba.ar

Índice

1. Introducción

El presente informe reúne la documentación del progreso del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación que busca permitir aprender los conceptos básicos de programación armando algoritmos utilizando bloques visuales.

2. Supuestos

- Al eliminar un bloque del algoritmo que se esta armando, este elimina todos los bloques subsiguientes.
- Un "algoritmo" puede tener infinitos niveles de bloques compuestos(llámese bloque repetir, bloque invertir, entre otros).
- Un "algoritmo" guardado no puede ser modificado una vez se ha guardado.Como consecuencia una vez guardado, el algoritmo que se estaba modificando se "pierde" y solo se puede obtener como un bloque no modificable.
- Un bloque de repetición repite el algoritmo contenido como si se tratase de un "for loop". Es decir repite secuencialmente los bloques dentro de él.
- Al moverse de una posición con el lápiz apoyado, solo se pintara la posición que se abandono.
- Todos los movimientos que realiza un personaje son consecutivos; no "salta" entre posiciones no adyacentes entre si.

3. Diagramas de clase

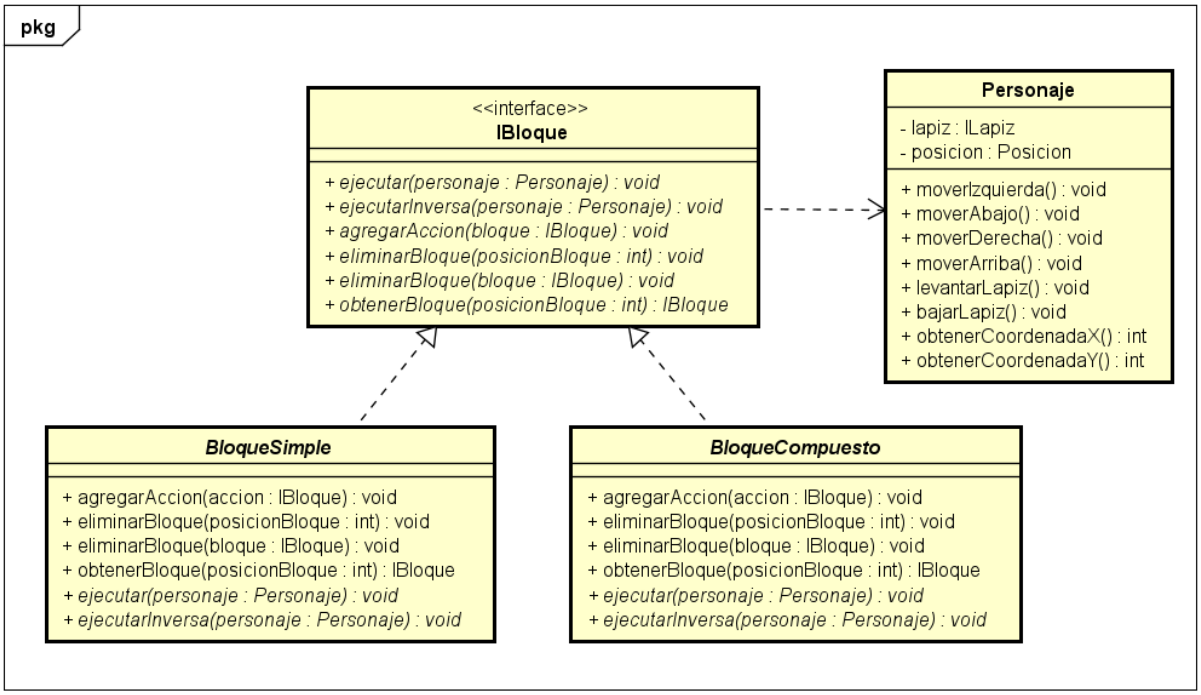

Figura 1: Diagrama Estructura de Bloques

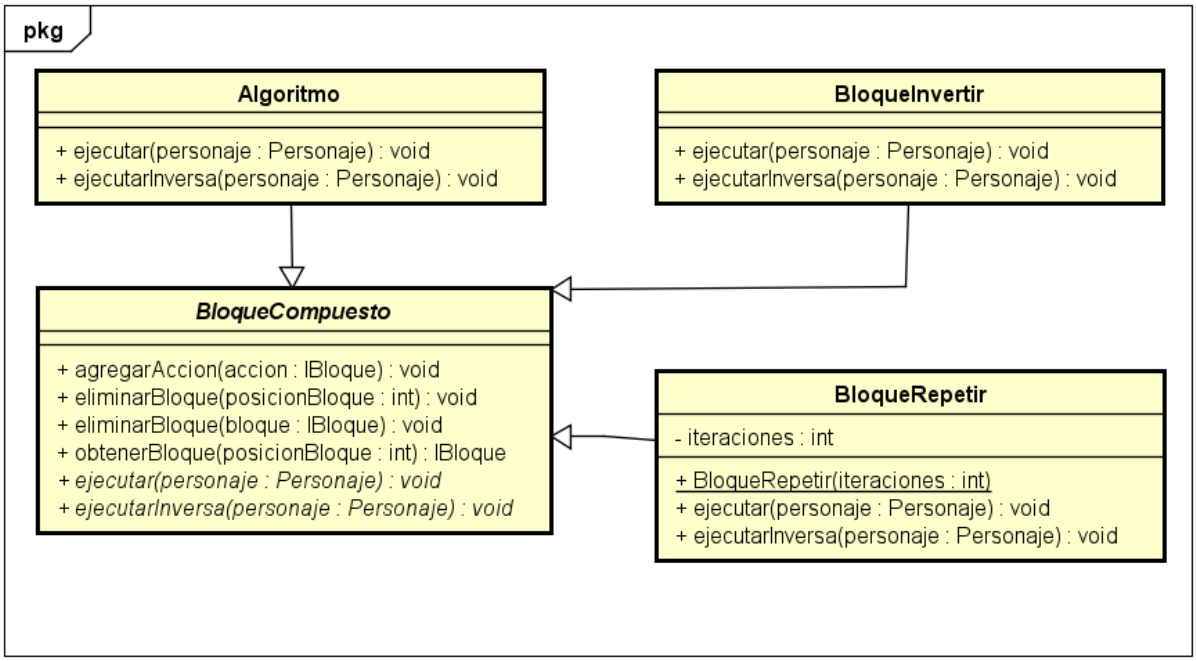

Figura 2: Bloques Compuestos disponibles en el modelo.

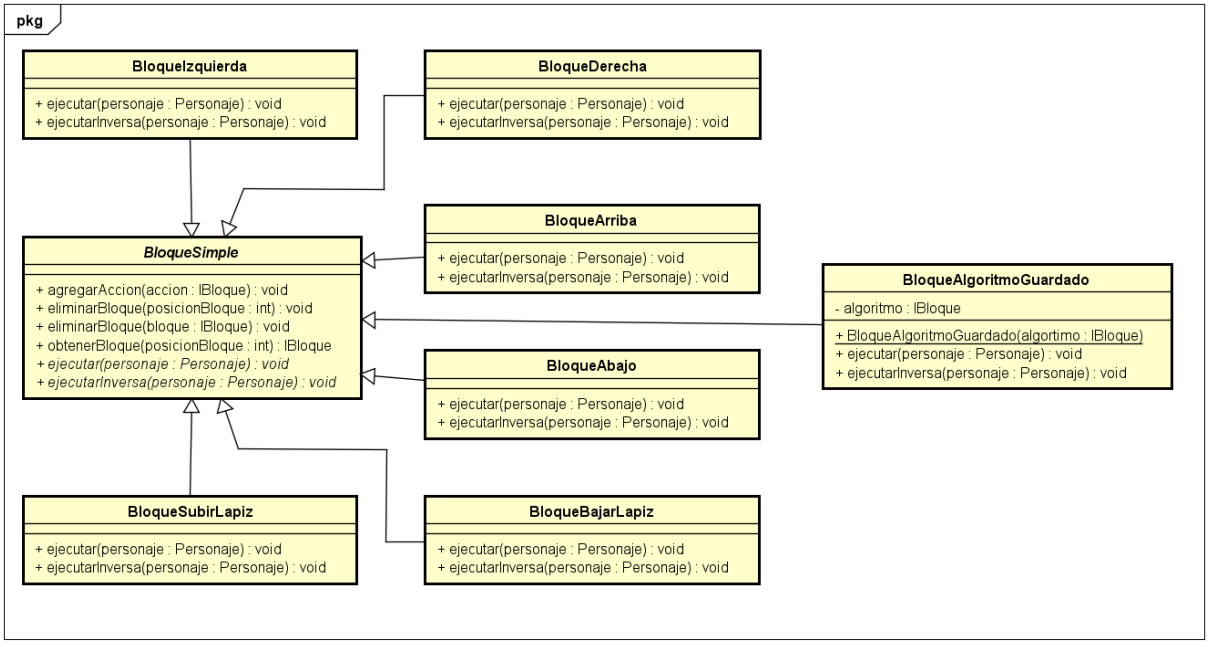

Figura 3: Bloques Simples disponibles en el modelo.

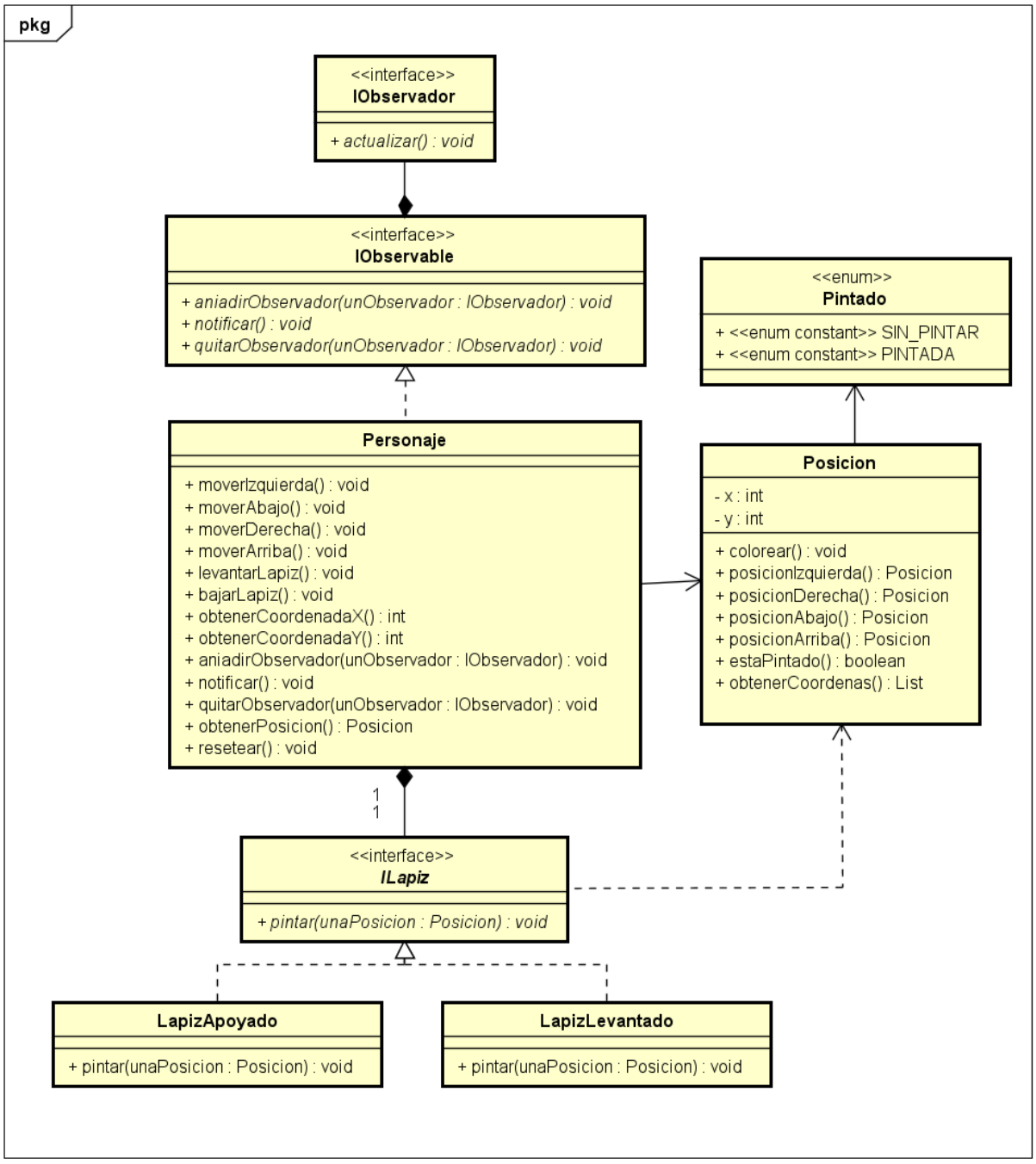

Figura 4: Relación Personaje con Posición.Un personaje es un observable.

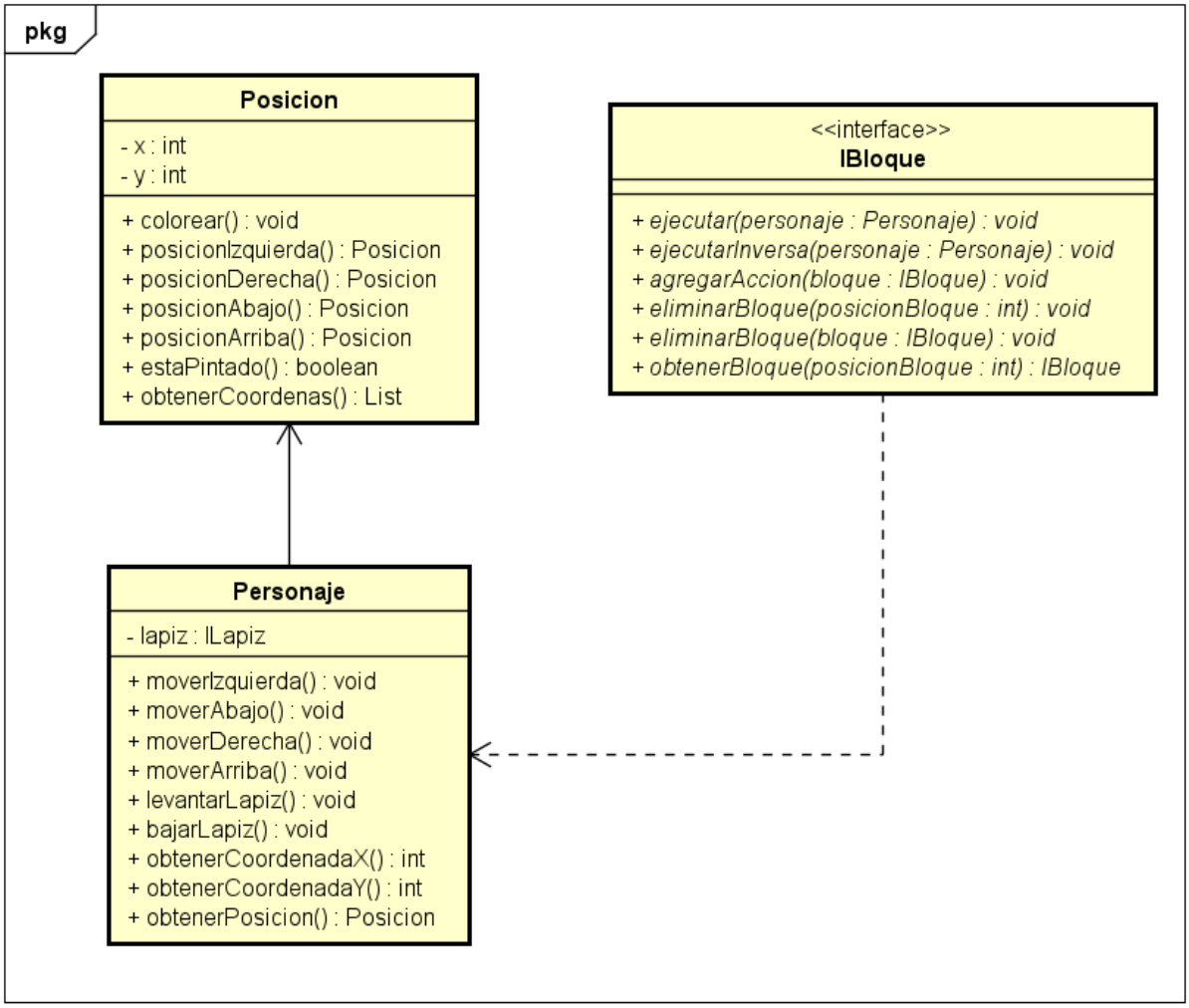

Figura 5: Relación entre personaje, bloque y posición.

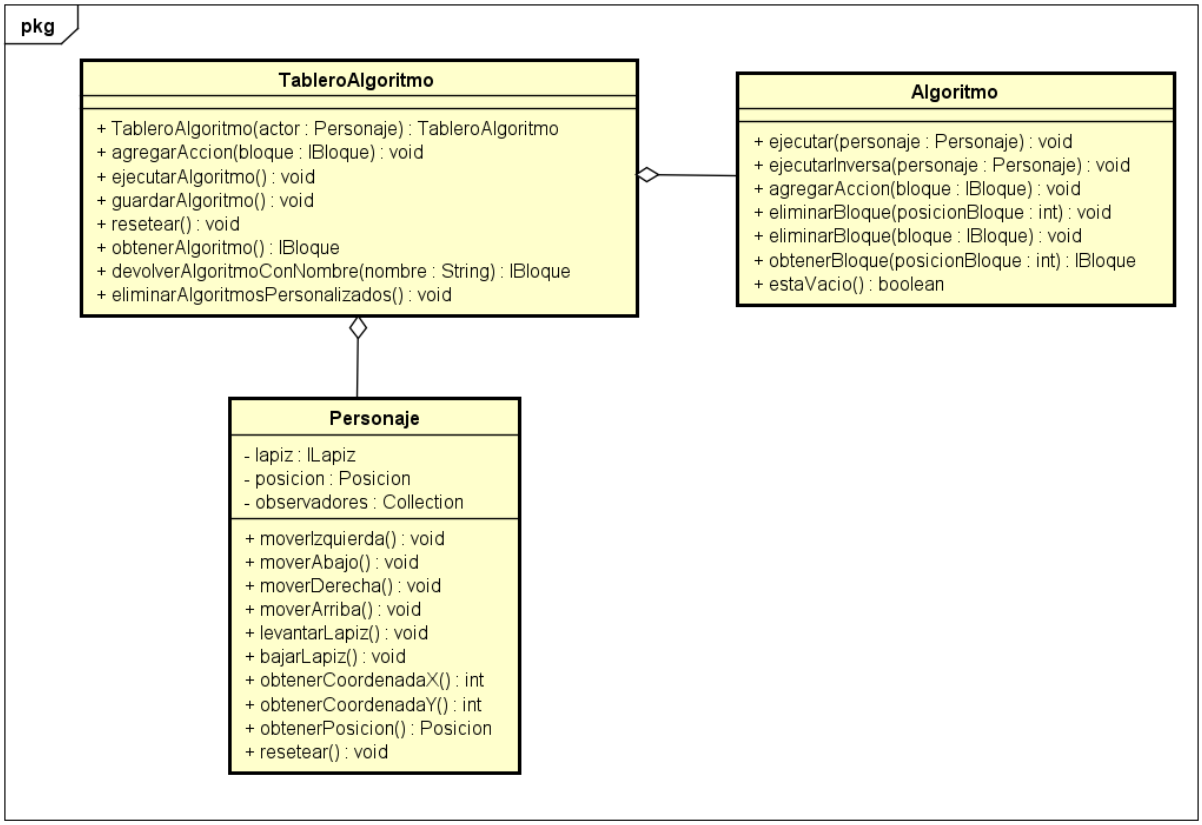

Figura 6: Tablero Algoritmo como interfaz al usuario.

4. Detalles de implementación

Para la implementación del modelo se propuso una clase "principal" **Personaje** que modela el objeto a mover mediante el algoritmo armado por bloques.
La posición de un personaje esta modela mediante la clase **Posición**. Esta clase se encarga de saber unas coordenadas y conseguir la siguiente posición según se pida. Cuando un personaje cambia de posición este puede pintar, o no, la posición en la que se encontraba; esto dependerá del estado del **Lápiz** al momento de moverse, pudiendo estar levantado o apoyado, pintando, o no, la posición abandonada según corresponda. Para implementar este comportamiento se hizo uso del patrón **State**.
Las ordenes de movimiento, así como la de cambiar el estado del lápiz, están dadas por los **Bloques**. Estos conocen al personaje, y le dicen las acciones que debe realizar mediante el uso de la interfaz publica del personaje. Si bien todos los grupos cumplen con la misma interfaz, estos están en divididos en dos grupos: los **Bloques Simples**, y los **Bloques Compuestos**.
Los bloques simples son aquellos que por su comportamiento, no envían mensajes a otros bloques sino que solamente al personaje.
Los bloques compuestos son aquellos que pueden, y para tener comportamiento deben, tener almacenado otros bloques.
Si bien ambas responden a la misma interfaz de Bloque, los bloques simples no responden a los comandos que intentar modificar el bloque, estos métodos lanzan excepciones. Esto es necesario para que el patrón de diseño usado **Composite**, sea posible de utilizar sin perder la posibilidad de modificar los bloques compuestos mas internos de un algoritmo.
Para notificar un cambio de posición a cualquier objeto interesado, un Personaje implementa la interfaz de **Observer**, notificando a cualquier objeto que este subscrito cuando este cambia de posición. Esto se utilizo luego para poder dibujar el personaje en su perteneciente tablero en la vista. Un **TableroAlgoritmo** es el responsable del manejo de un Algoritmo principal(armar, ejecutar, guardar) y entregárselo al personaje correspondiente a ejecutar dicho algoritmo.

5. Diagramas de secuencia

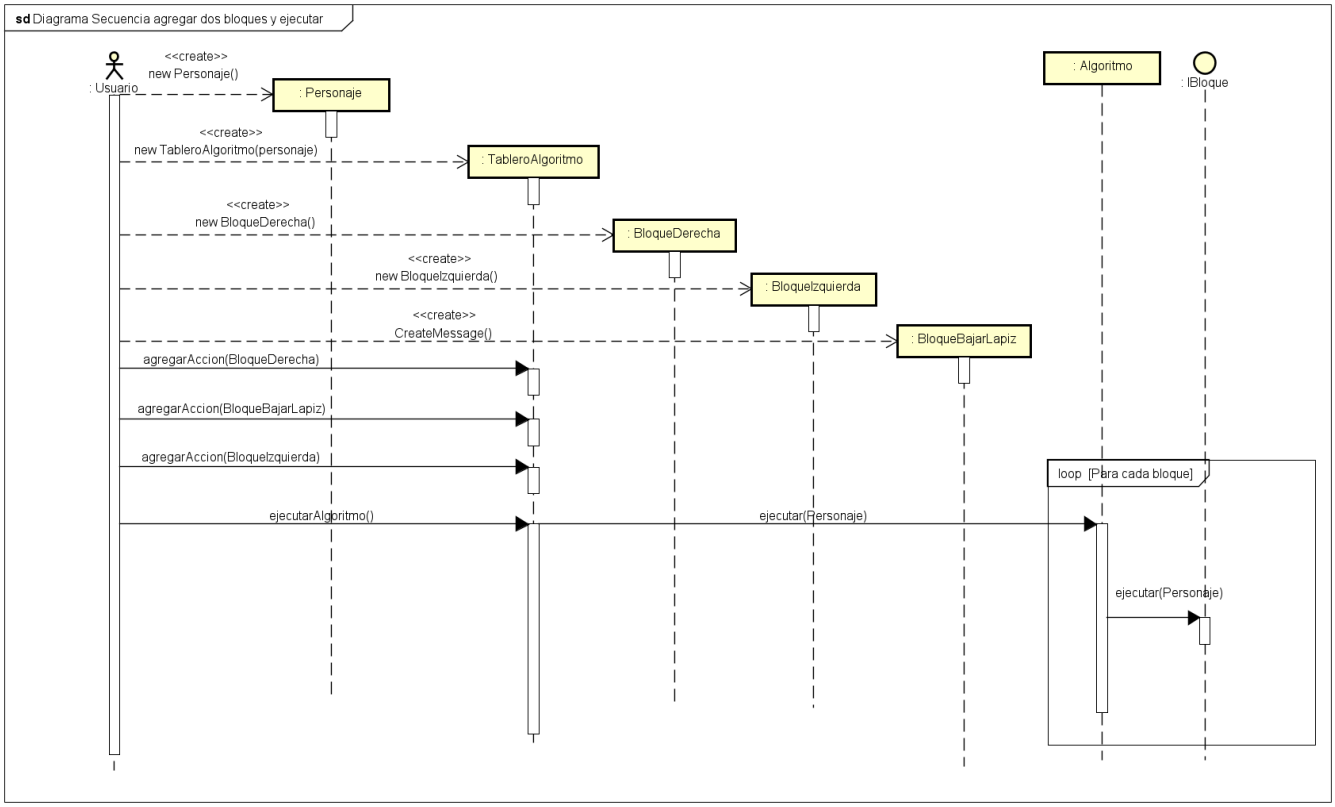

Figura 7: Ejecutar algoritmo con tres bloques.

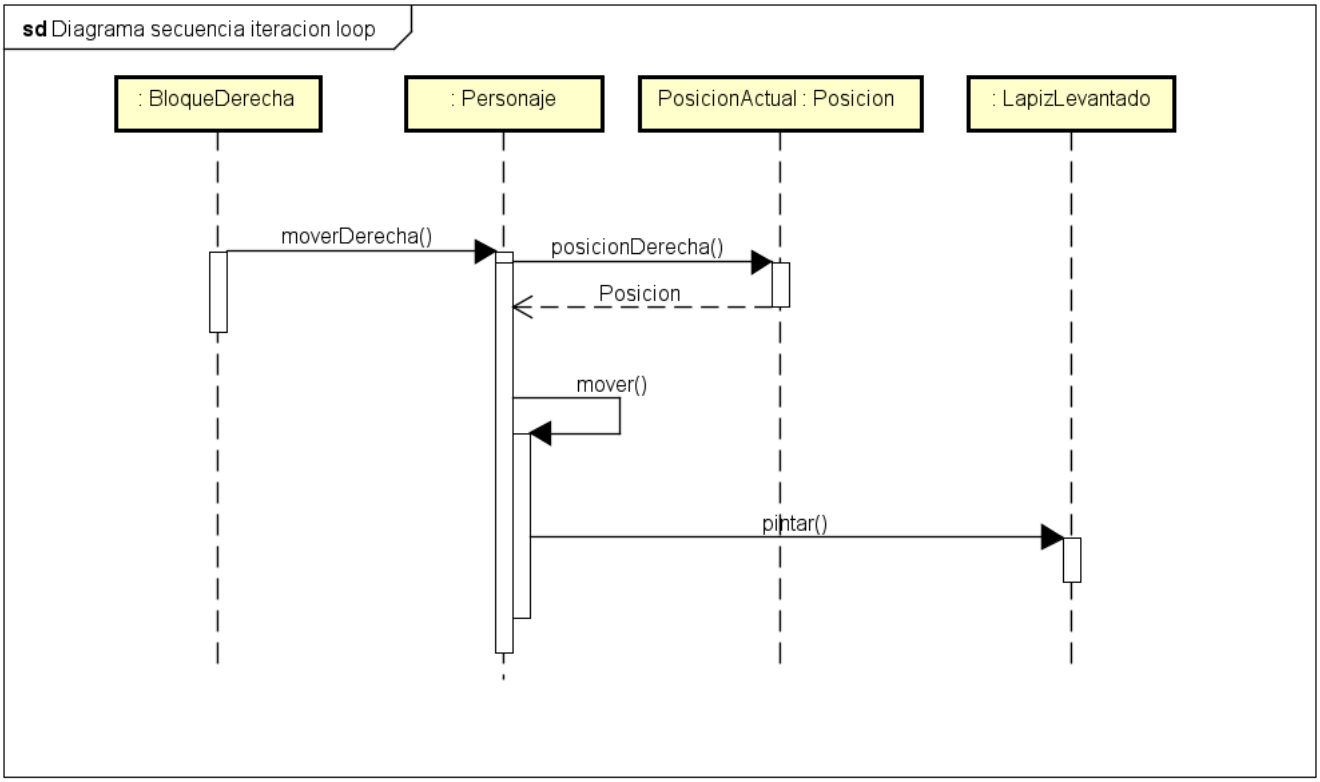

Figura 8: Iteracion del loop de la fiugra 7.

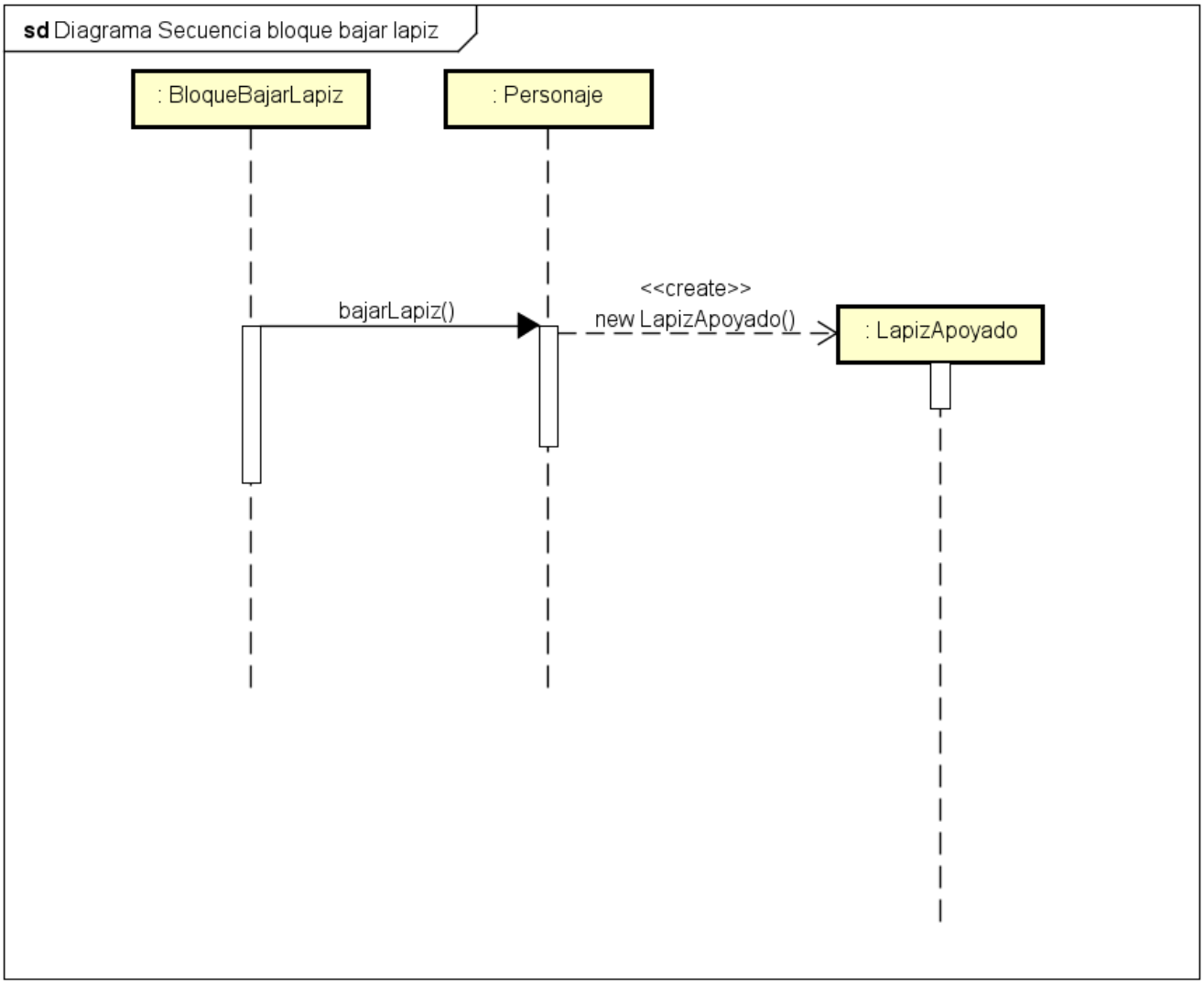

Figura 9: Iteracion del loop de la figura 7 , se instancia un nuevo lapiz apoyado.

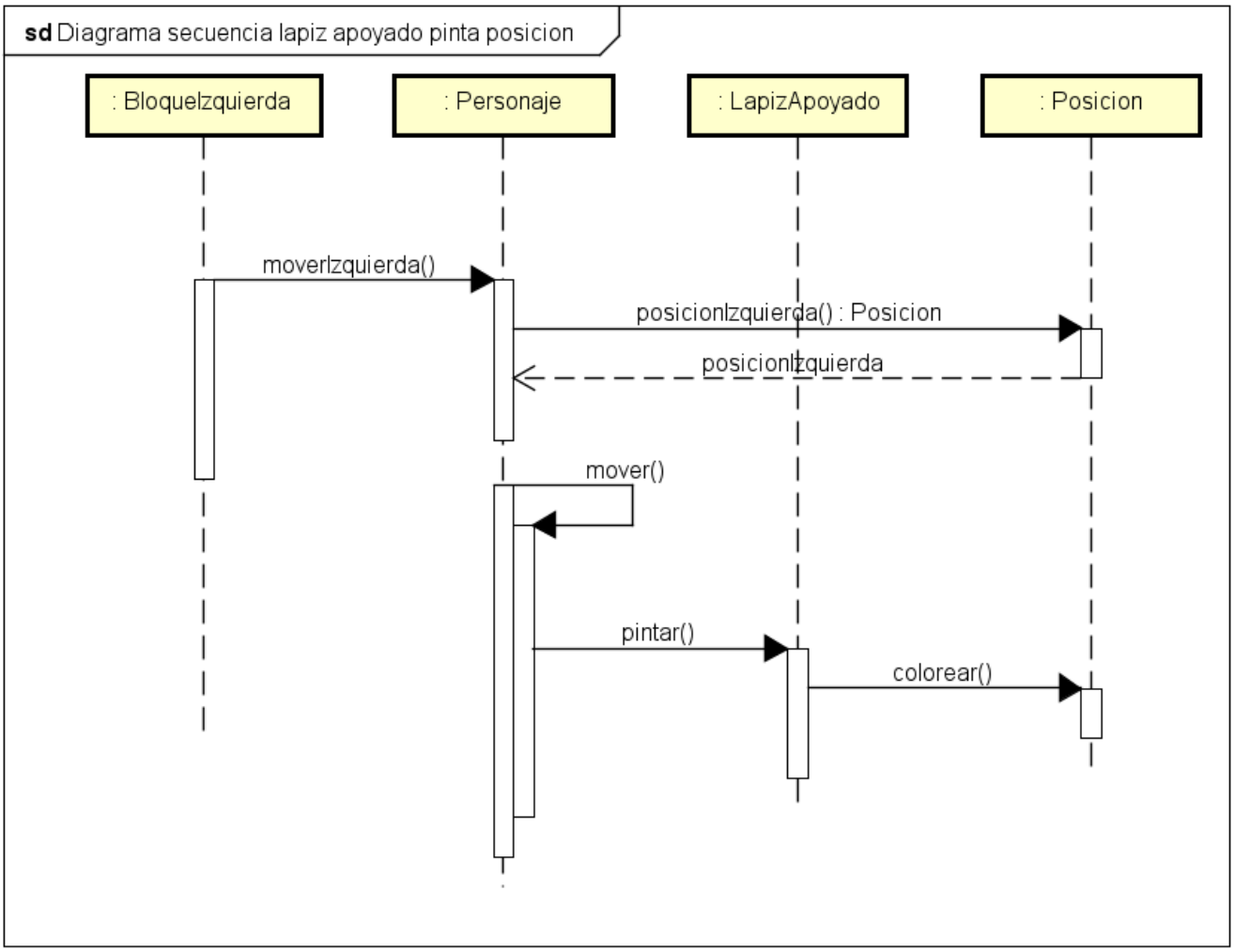

Figura 10: Iteracion del loop de la figura 7 , ahora se pinta la posicion.

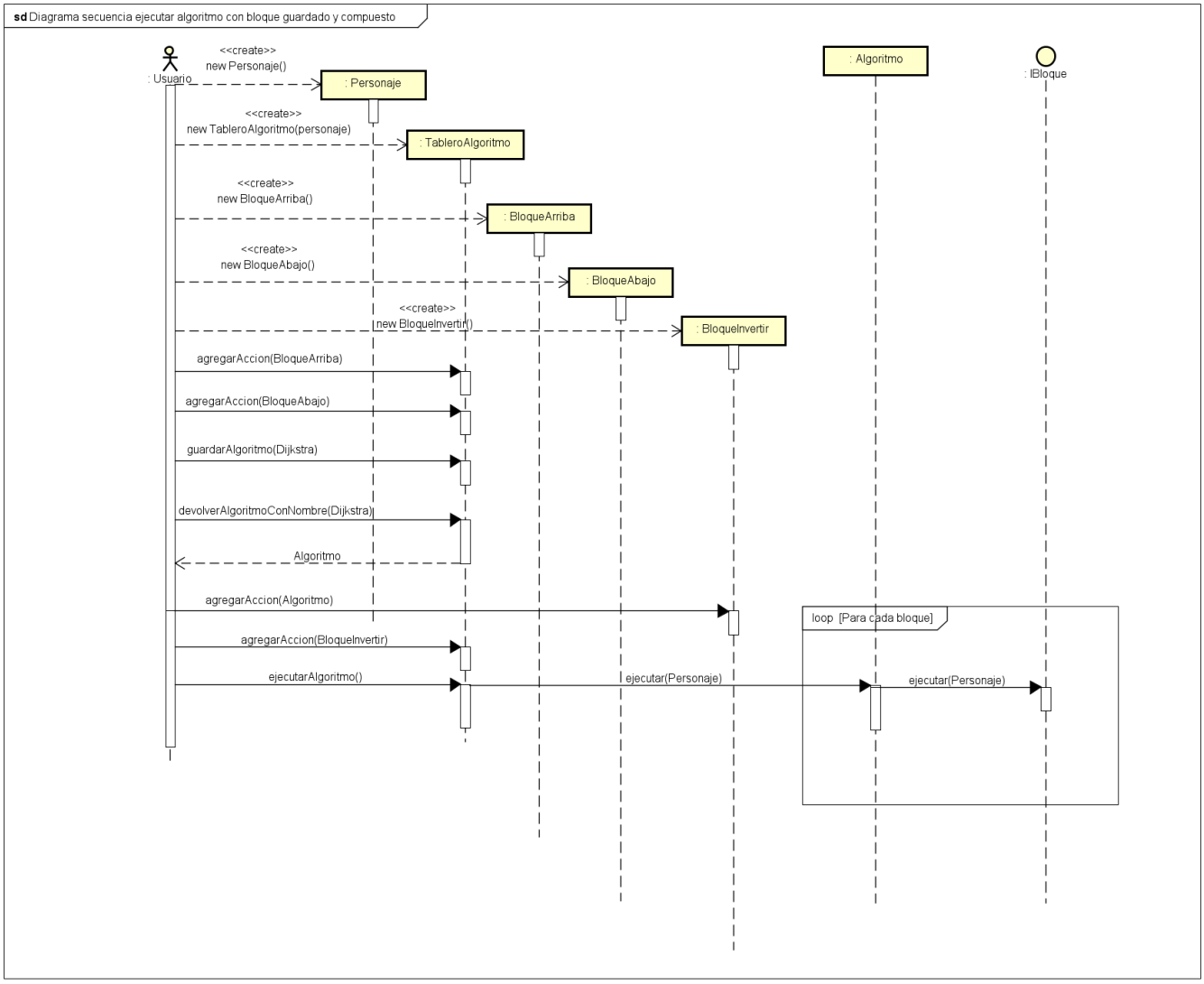

Figura 11: Ejecutar algoritmo con bloque guardado y compuesto.

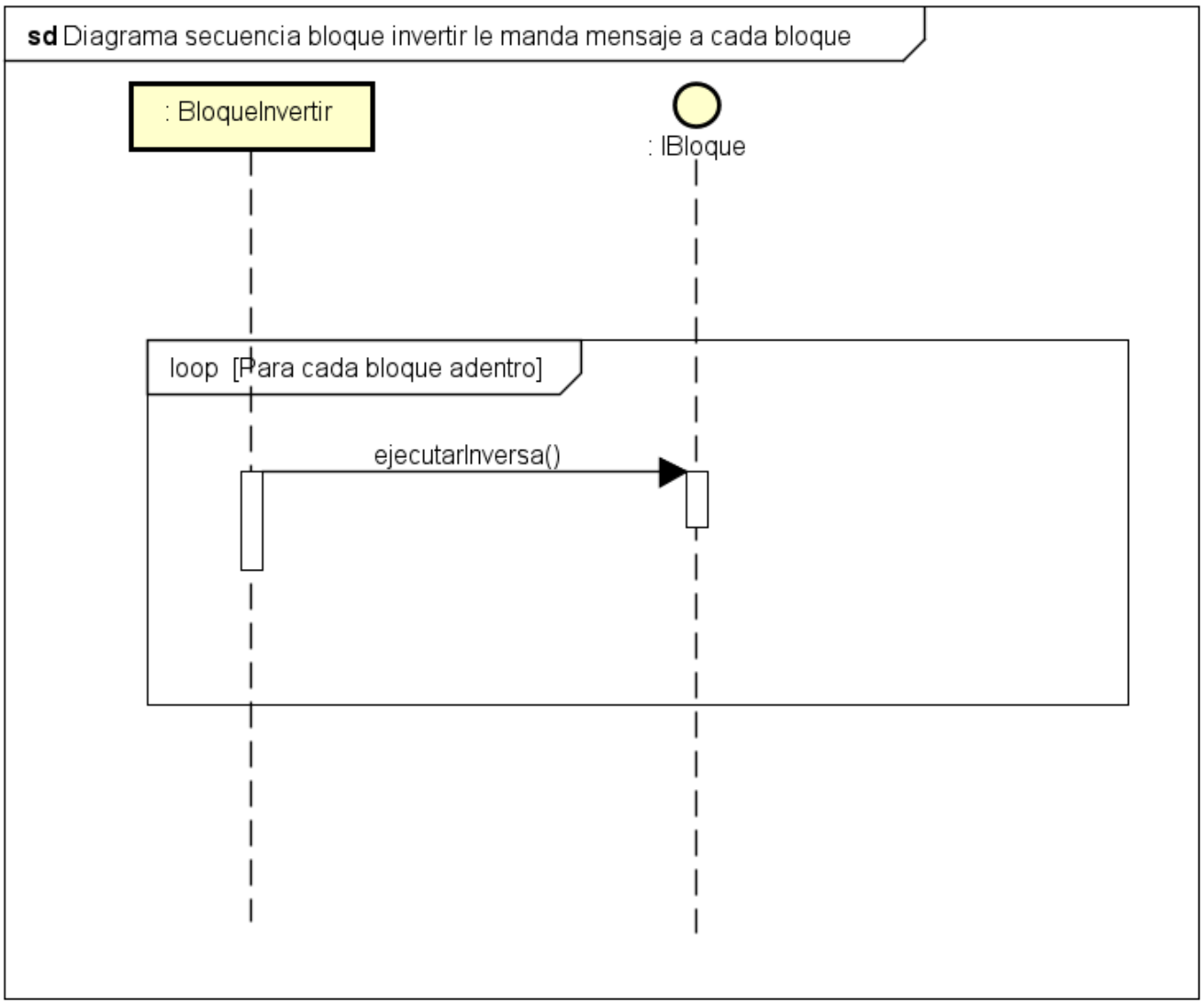

Figura 12: Iteración del loop de la figura 11.

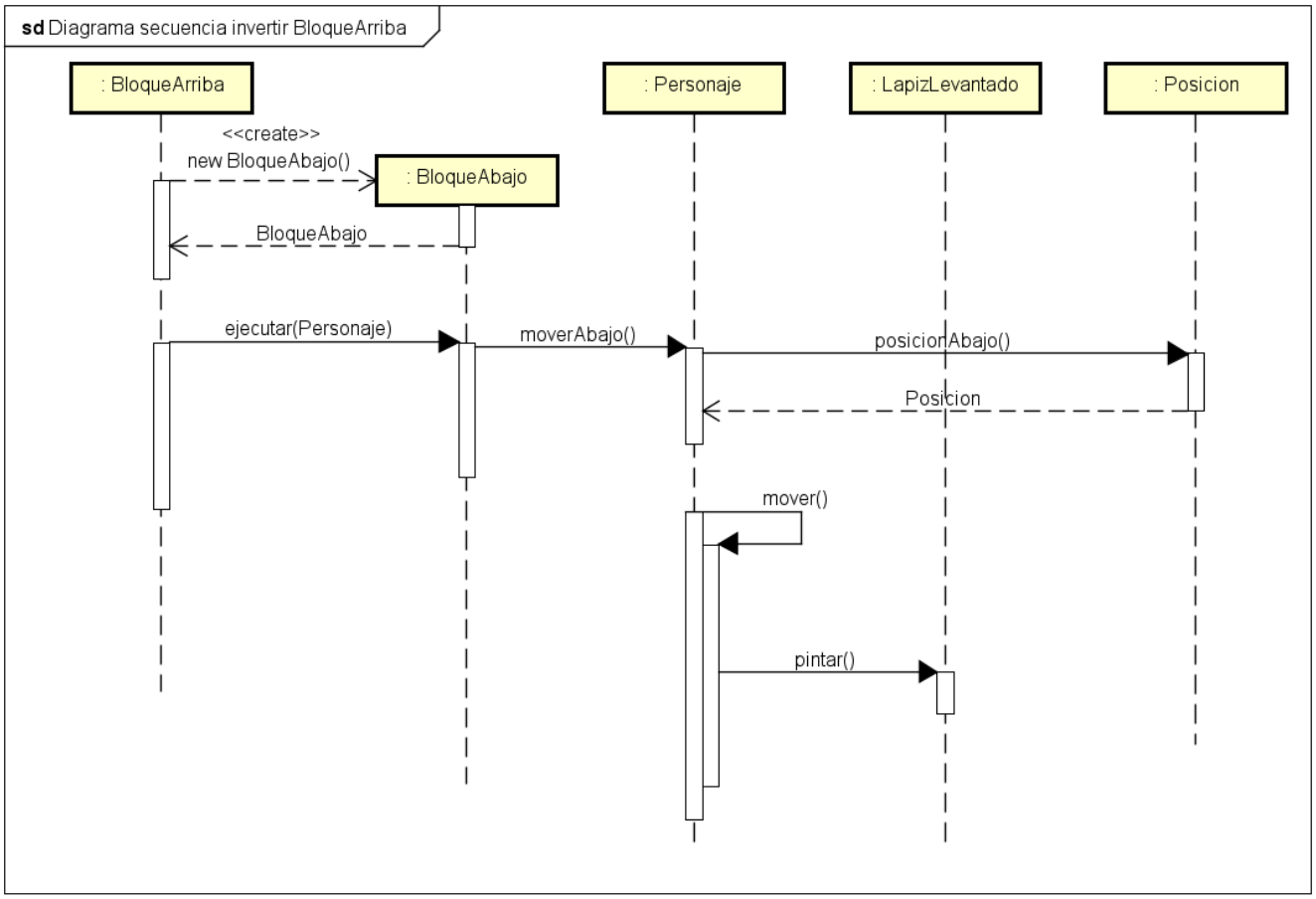

Figura 13: Bloque arriba instancia un bloque abajo y lo ejecuta.

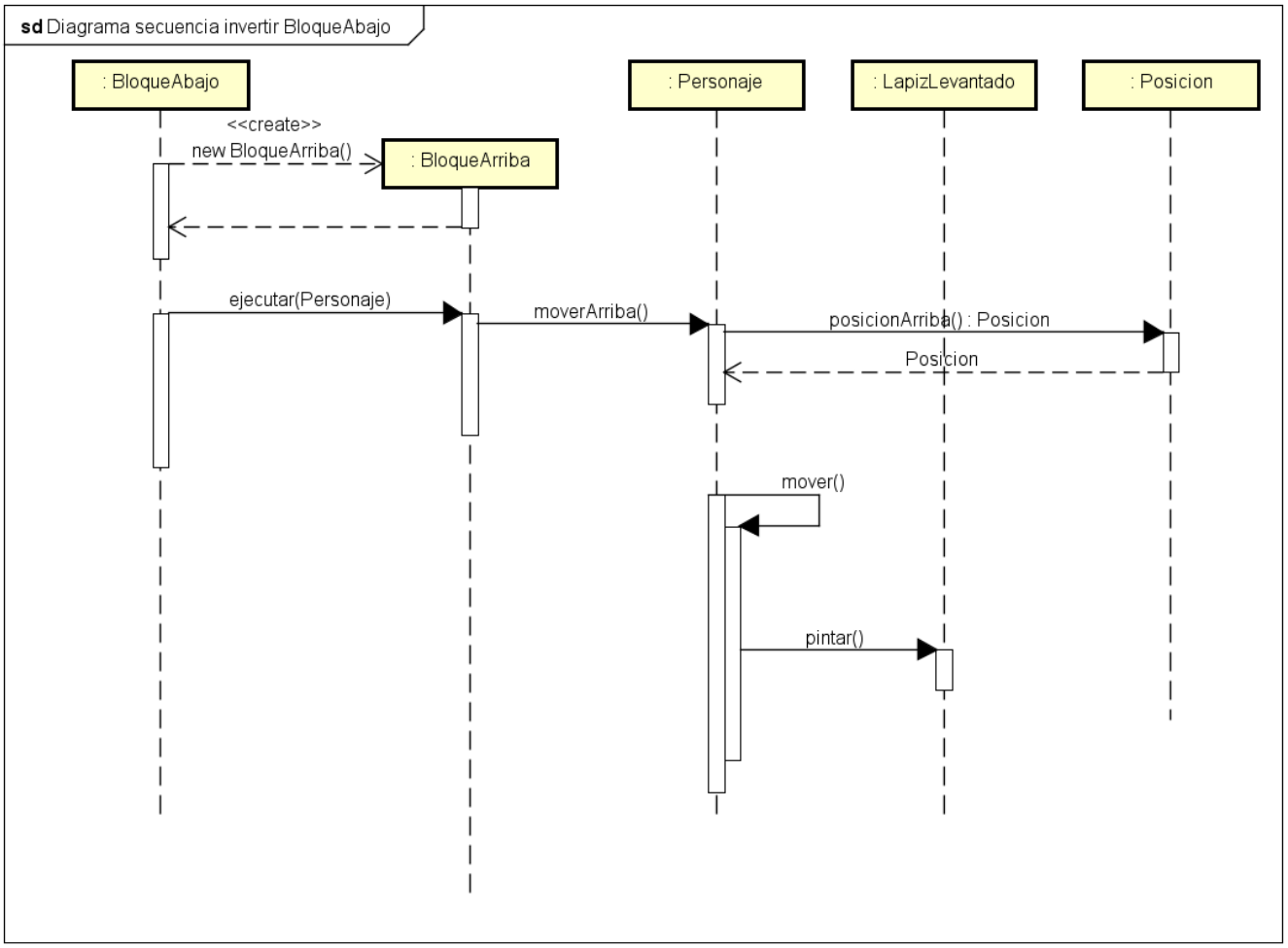

Figura 14: Bloque arriba instancia un bloque arriba y lo ejecuta.

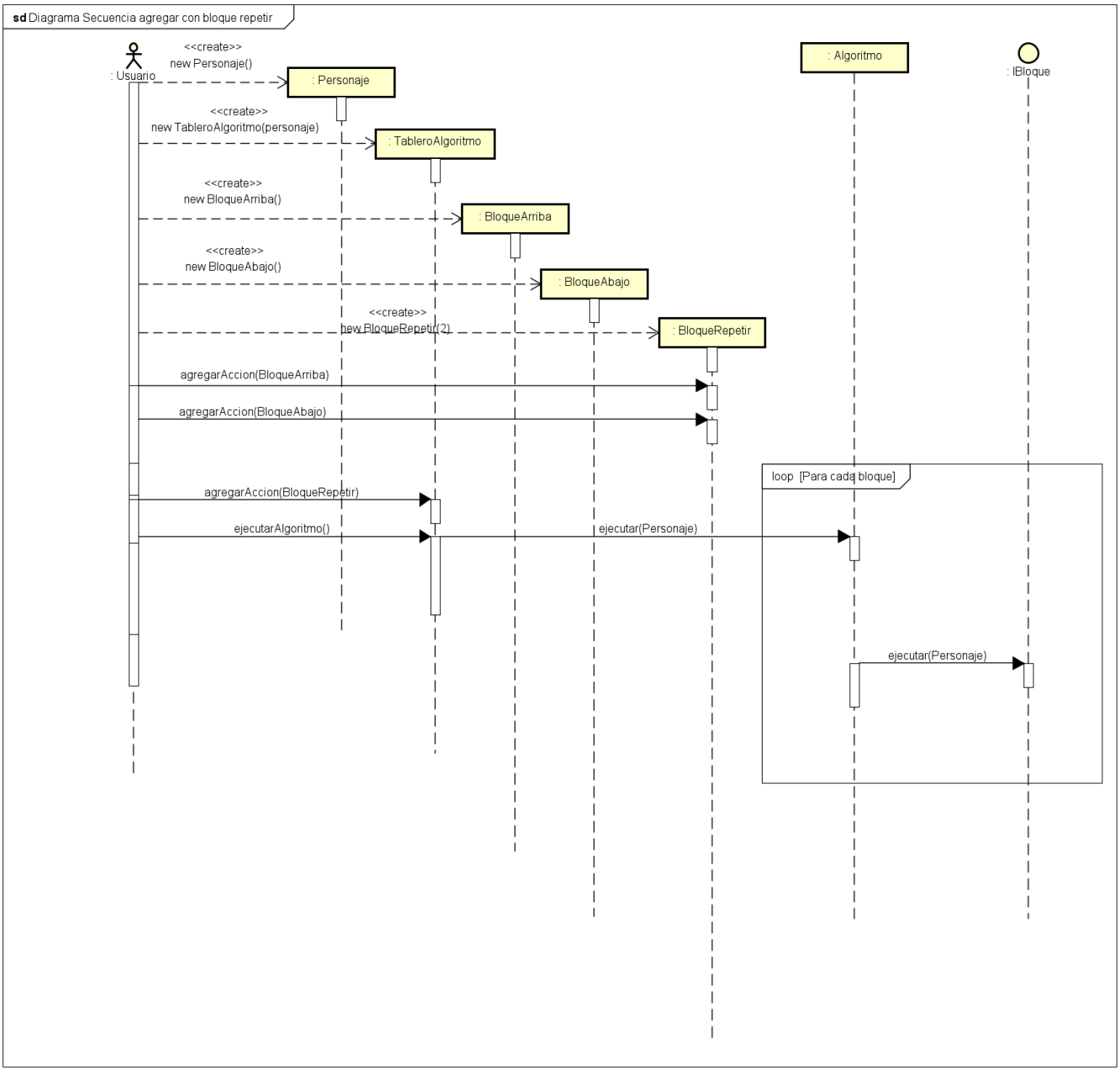

Figura 15: Ejecutar algoritmo con bloque repetir dos veces.

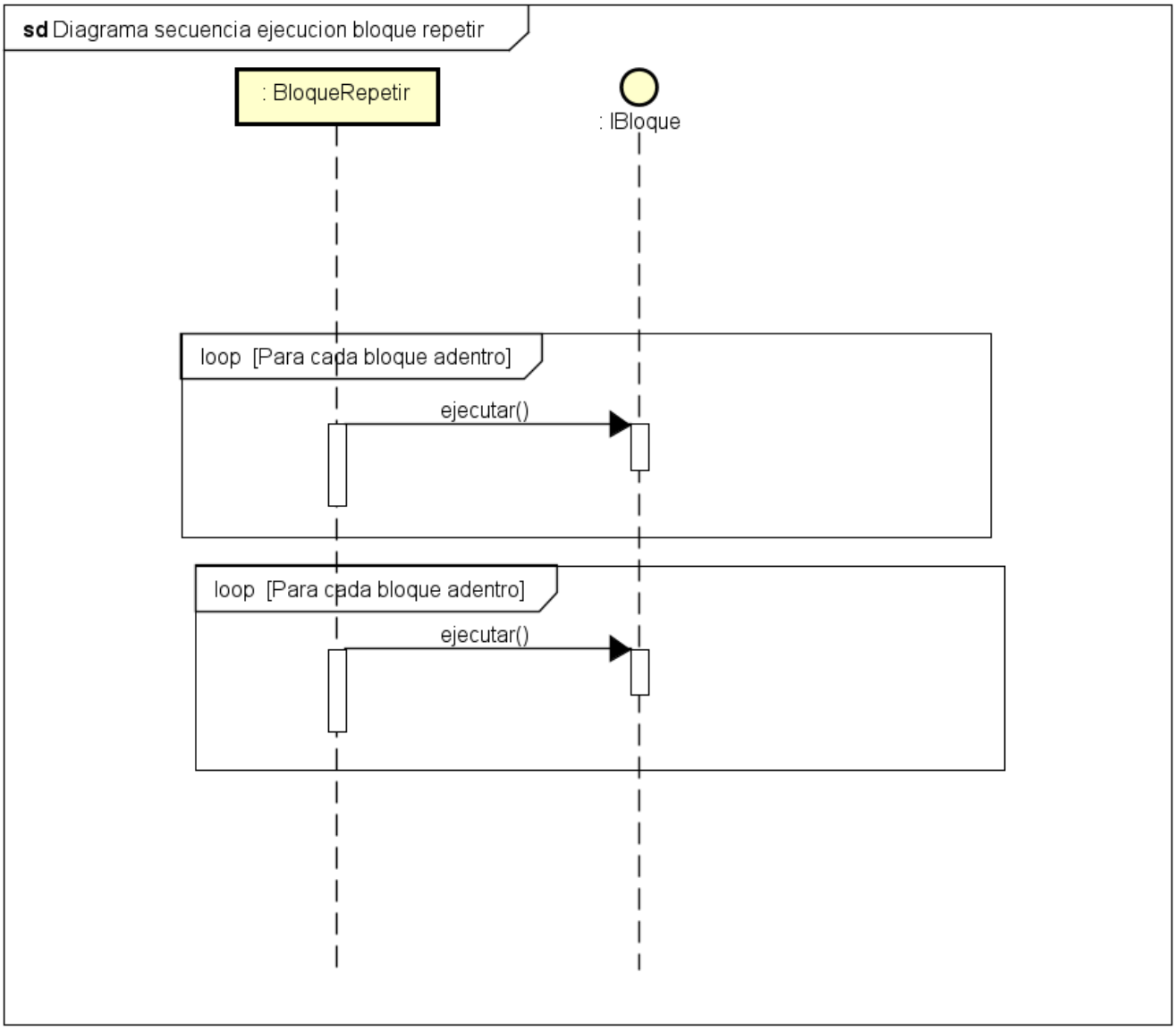

Figura 16: Bloque repetir ejecuta dos veces los bloques que contiene.

6. Diagrama de Paquetes

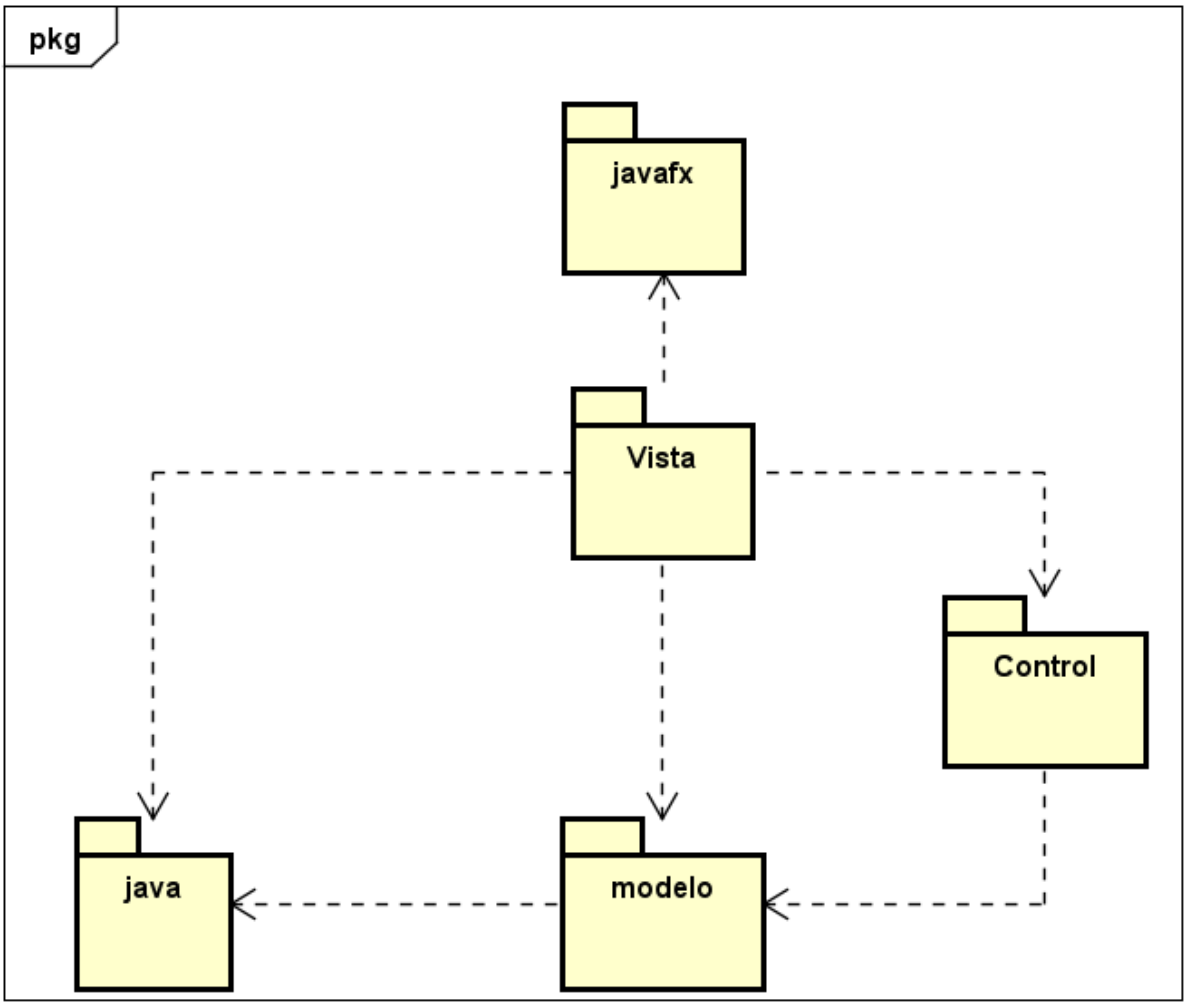

Figura 17: Diagrama de dependencia de los paquetes del programa.

En el diagrama de paquetes se puede apreciar que la parte del modelo solo depende de los paquetes de java. En otras palabras, el modelo no depende de la vista ni de sus controladores; cualquier cambio que se realiza a la interfaz del usuario, como a la lógica que afecta a esta, no afecta al modelo interno.
El diagrama es una simplificación de un diagrama generado con la herramienta **AstahUML** sobre todas las clases del programa, para verificar que el modelo efectivamente no tenga ningún acoplamiento con la vista.

7. Diagrama de Estados

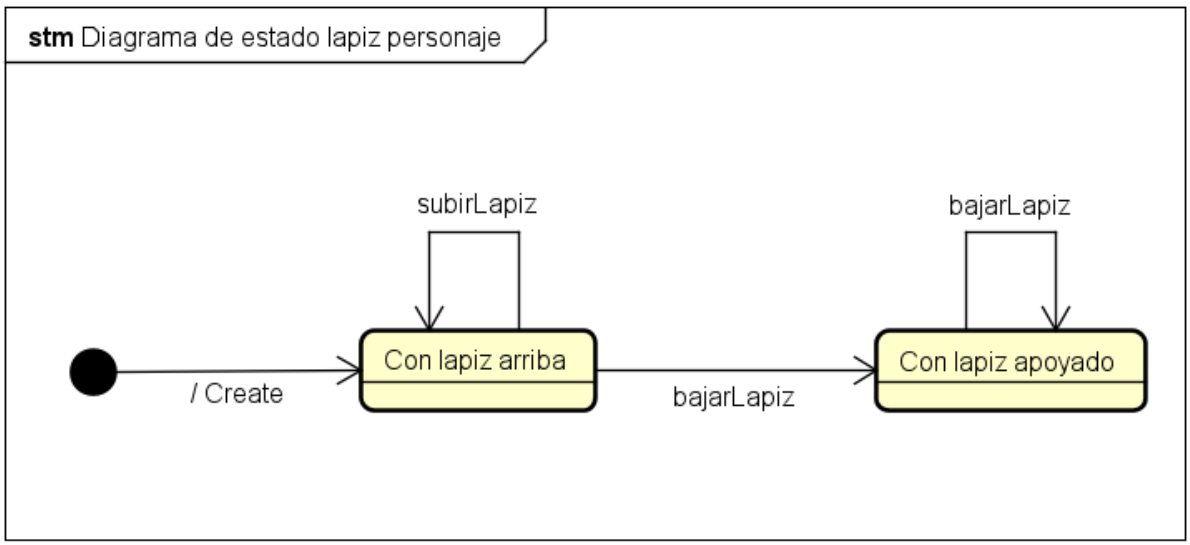

Figura 18: Diagrama de estados personaje.

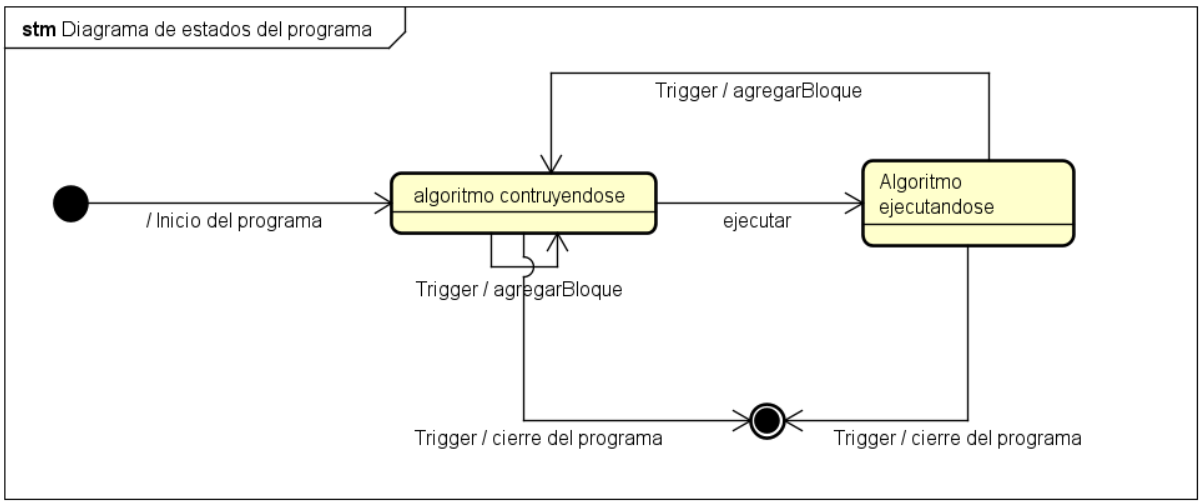

Figura 19: Diagrama de estados del progama.

8. Excepciones

- **BloqueFueraDeRangoExcepcion:** Esta excepción se lanza en caso de querer eliminar u obtener un bloque de un bloque compuesto y que la posición se encuentre fuera de rango. Su tratamiento no sera necesario desde los controladores utilizados.
- **BloqueNoEncontradoExcepcion:** Esta excepcion se lanza en caso de que al querer eliminar un bloque de un bloque compuesto, este no se encuentre dentro del mismo. Su tratamiento no sera necesario desde los controladores utilizados.
- **OperacionInvalidaBloqueSimpleExcepcion:** Esta excepcion se lanza en caso de querer realizar la acción de agregar o eliminar bloques a un bloque simple. Su tratamiento no sera necesario desde los controladores utilizados.
- **ElNombreDeUnAlgoritmoNoPuedeEstarVacioExcepcion:** Esta excepcion se lanza en caso de querer guardar un algoritmo sin un nombre. Su tratamiento desde un controlador generará una alerta en la vista para advertirle al usuario.
- **NoHayBloquesEnElAlgoritmoAGuardarExcepcion:** Esta excepcion se lanza en caso de que no haya ningun bloque en el algoritmo que se quiere guardar. Su tratamiento desde un controlador generará una alerta en la vista para advertirle al usuario.
- **NoHayAlgoritmoGuardadoExcepcion:** Esta excepcion se lanza en caso de que no exista un algoritmo con el nombre pedido en la clase TableroAlgoritmo. Su tratamiento no sera necesario desde los controladores utilizados.