FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Java

[7507/9502] Algoritmos y Programación III
Curso 2
Segundo cuatrimestre de 2020

Alumno 1 :	Araujo, Luis
Número de padrón: Email:	102112 laraujo@fi.uba.ar
Alumno 2 :	Arroyo, Sebastian
Número de padrón: Email:	100728 sarroyo@fi.uba.ar
Alumno 3 :	San Martin, Nicolas
Número de padrón: Email:	104320 nsanmartin@fi.uba.ar
Alumno 4 :	Branko Tintilay Tacacho, Ivan
Número de padrón: Email:	102479 btintilay@fi.uba.ar

Índice

1. Introducción

En el presente informe se documentará la implementación del trabajo práctico 2 de la materia Algoritmos y Programación III, el cual consiste en desarrollar un juego orientado al aprendizaje de los conceptos básicos de la programación, mediante el armado de algoritmos usando una serie de bloques con distintas funcionalidades que se verán reflejadas a través de un dibujo en el tablero. El lenguaje de programación que se utilizó es Java, siguiendo el paradigma de la Programación Orientada a Objetos (POO).

2. Supuestos

En esta sección se explicitarán aquellas características del Juego que consideraremos esenciales para el desarrollo del programa, mismas las cuales no se encuentran explicadas en su totalidad por el enunciado del trabajo.

1. El personaje no podrá salirse de los limites del tablero, cuando este cruce un borde, cambiará su posición al lado opuesto de este mismo.
2. Podrá existir únicamente un personaje en el tablero.
3. Los bloques que se podrán invertir son los que tienen como funcionalidad el movimiento del personaje y los que cambian la posición del lápiz. Asumiendo también, la inversión de los bloques que fuesen contenidos por el bloque de repetición.
4. Al crearse un personaje este iniciará, por defecto, con el lápiz arriba.

3. Modelo de dominio

En cuanto al diseño implementado, se consideró como clase principal a Tablero, que es una interface de la que hacen uso las clases (que llamaremos sectores) SectorBloques, SectorDibujo y SectorAlgoritmo.

El primero se encargará de contener y mostrar los bloques con los que puede interactuar el usuario, que al seleccionarlos se enviarán y guardarán en SectorAlgoritmo, para que en el momento que el usuario ejecute el algoritmo, los bloques se comuniquen con el personaje, el cual a través de su movimiento y el uso del lápiz, podrá representar, o no, un dibujo en la posición en la que se encuentre, dependiendo de si el lápiz se encuentra hacia arriba o hacia abajo, viendo reflejado esto en SectorDibujo.

4. Diagramas de Clase

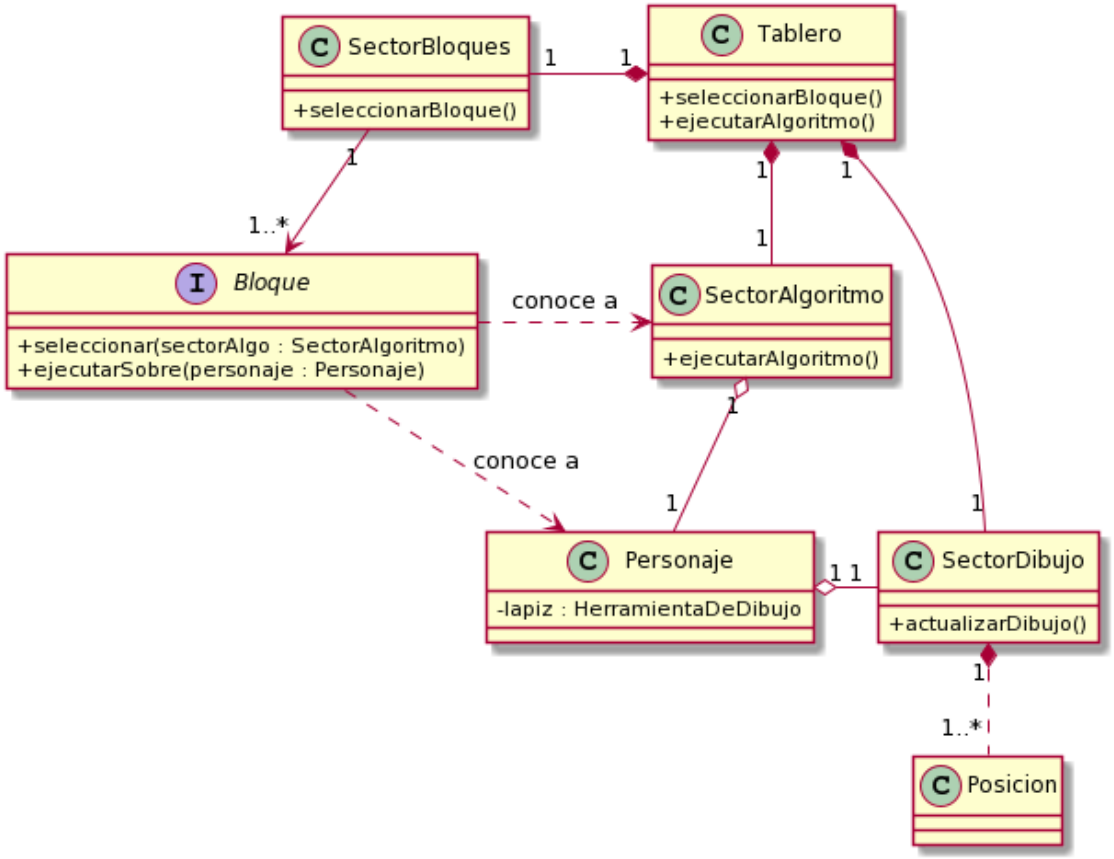

Figura 1: Diagrama principal del modelo a seguir, donde se muestra la Relación del Tablero con los Sectores Bloques, Algoritmo y Dibujo junto con el Personaje.

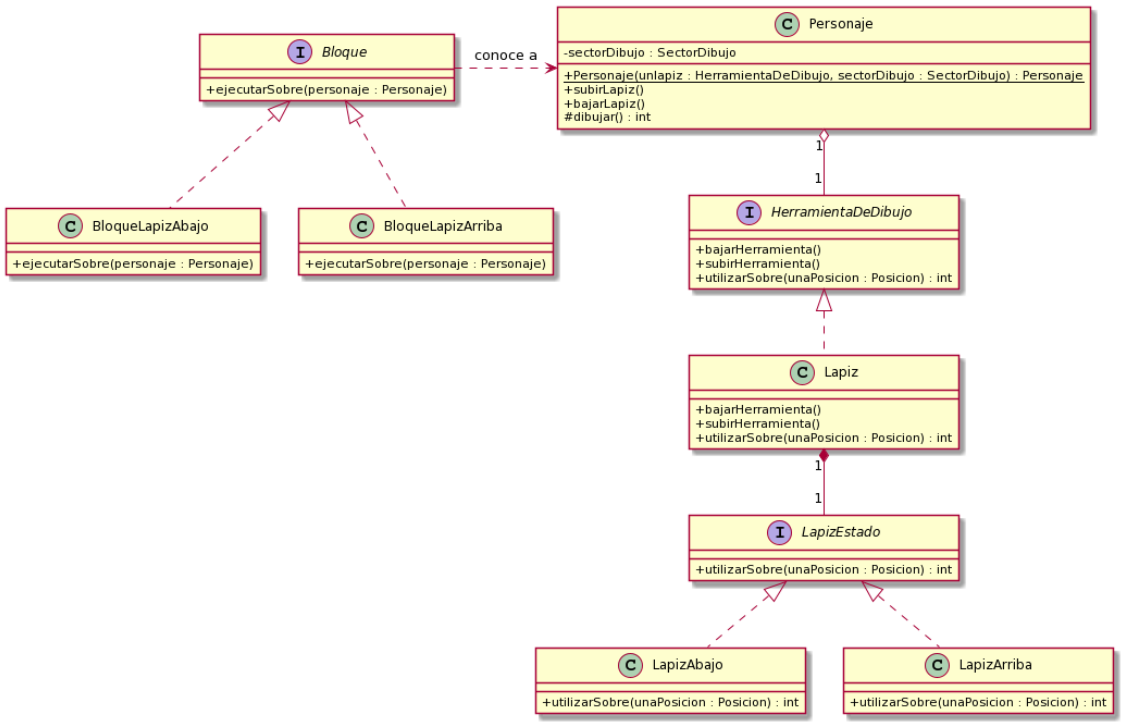

Figura 2: Diagrama sobre la relación de Personaje con la interfaz HerramientaDeDibujo y los estados que tiene Lapiz, LapizArriba y LapizAbajo, también se representan los bloques que suben y bajan el lápiz.

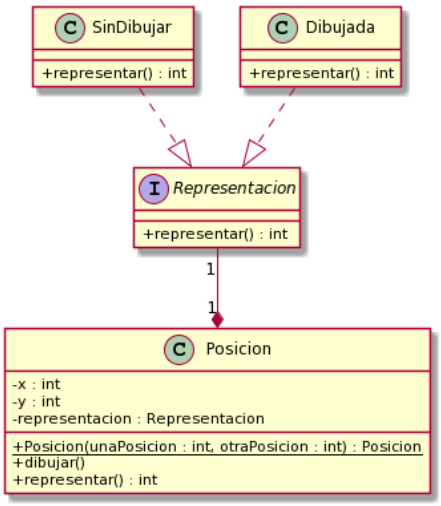

Figura 3: Diagrama sobre la relación de Posición con la interfaz Representación y las clases que la implementan Posición, SinDibujar y Dibujada.

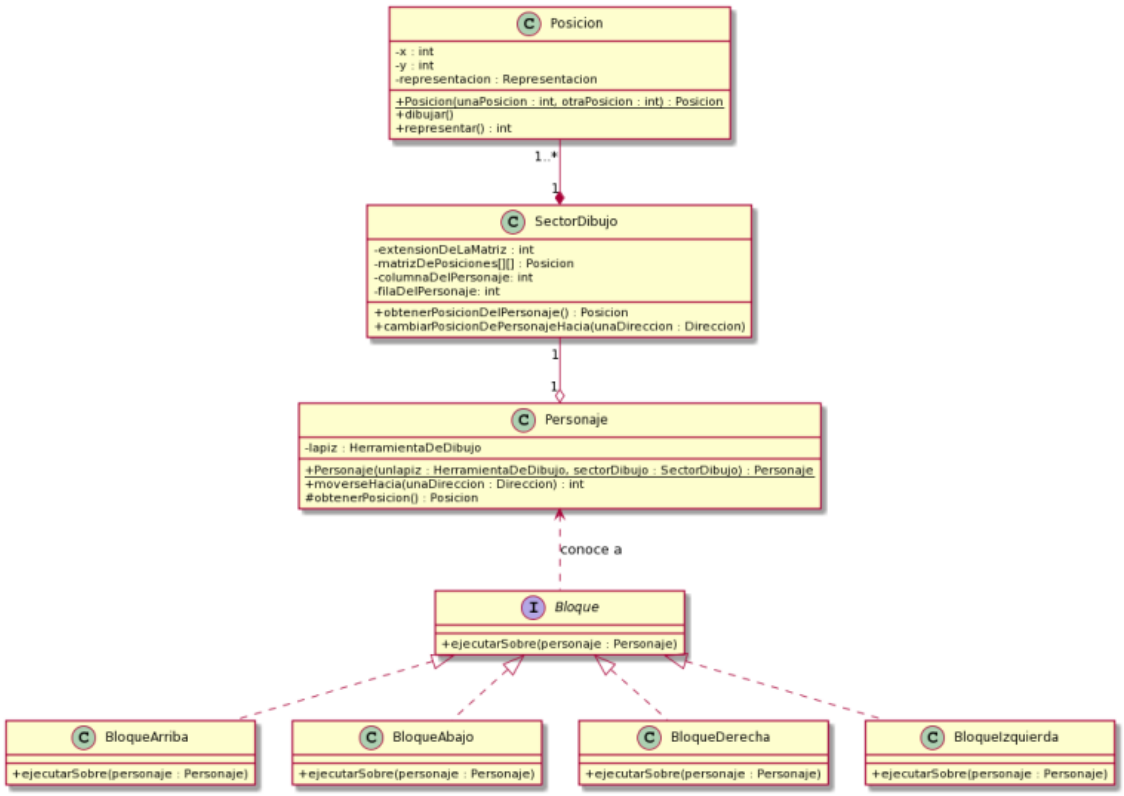

Figura 4: Diagrama sobre la relación de Personaje con la clase Posición, SectorDibujo y Bloque.

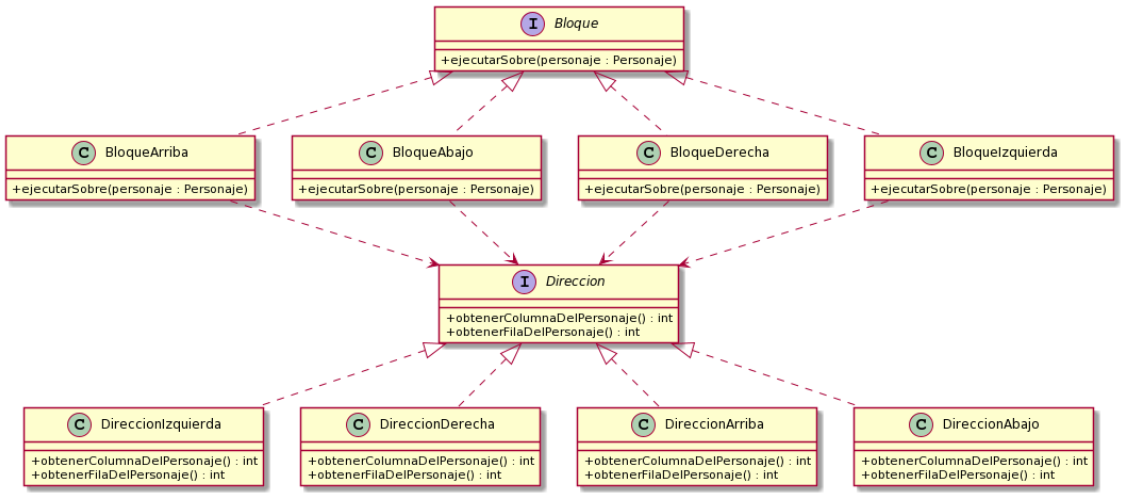

Figura 5: Diagrama sobre la relaciones entre los Bloques de movimiento y las direcciones que implementan la interfaz Dirección.

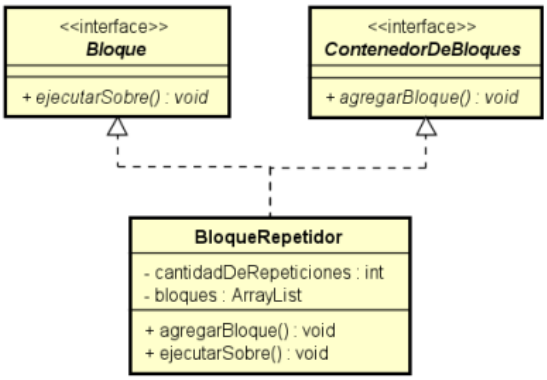

Figura 6: Diagrama sobre el Bloque Repetidor.

5. Diagramas de Secuencia

5.1. Diagramas de pruebas

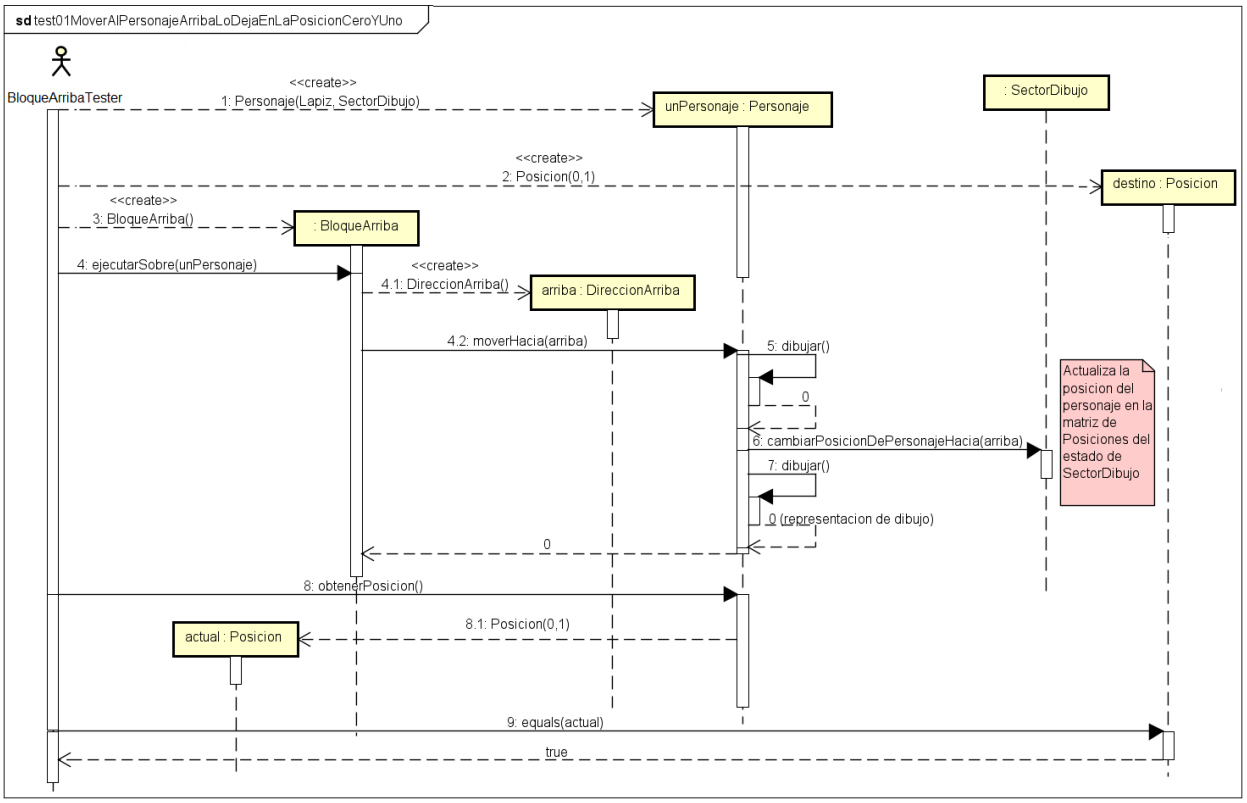

Figura 7: Diagrama sobre la ejecución de un algoritmo con un Bloque para mover arriba al Personaje.

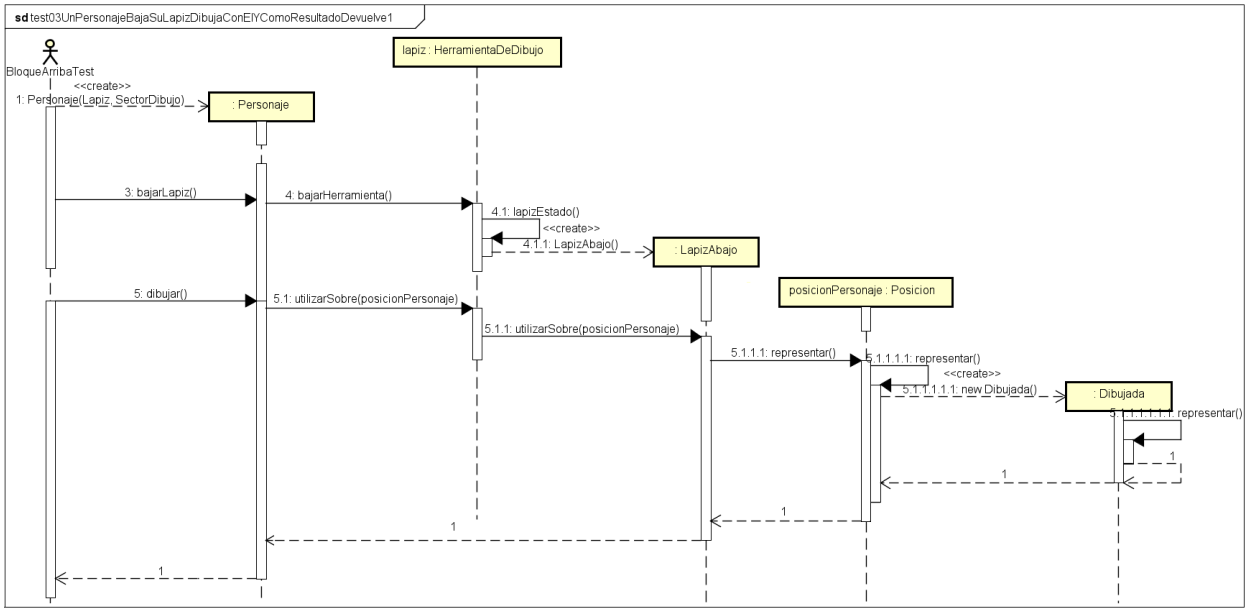

Figura 8: Diagrama sobre la ejecución de un algoritmo donde el Personaje baja su lápiz, dibujando con este mismo sobre la Posición en la que se encuentra.

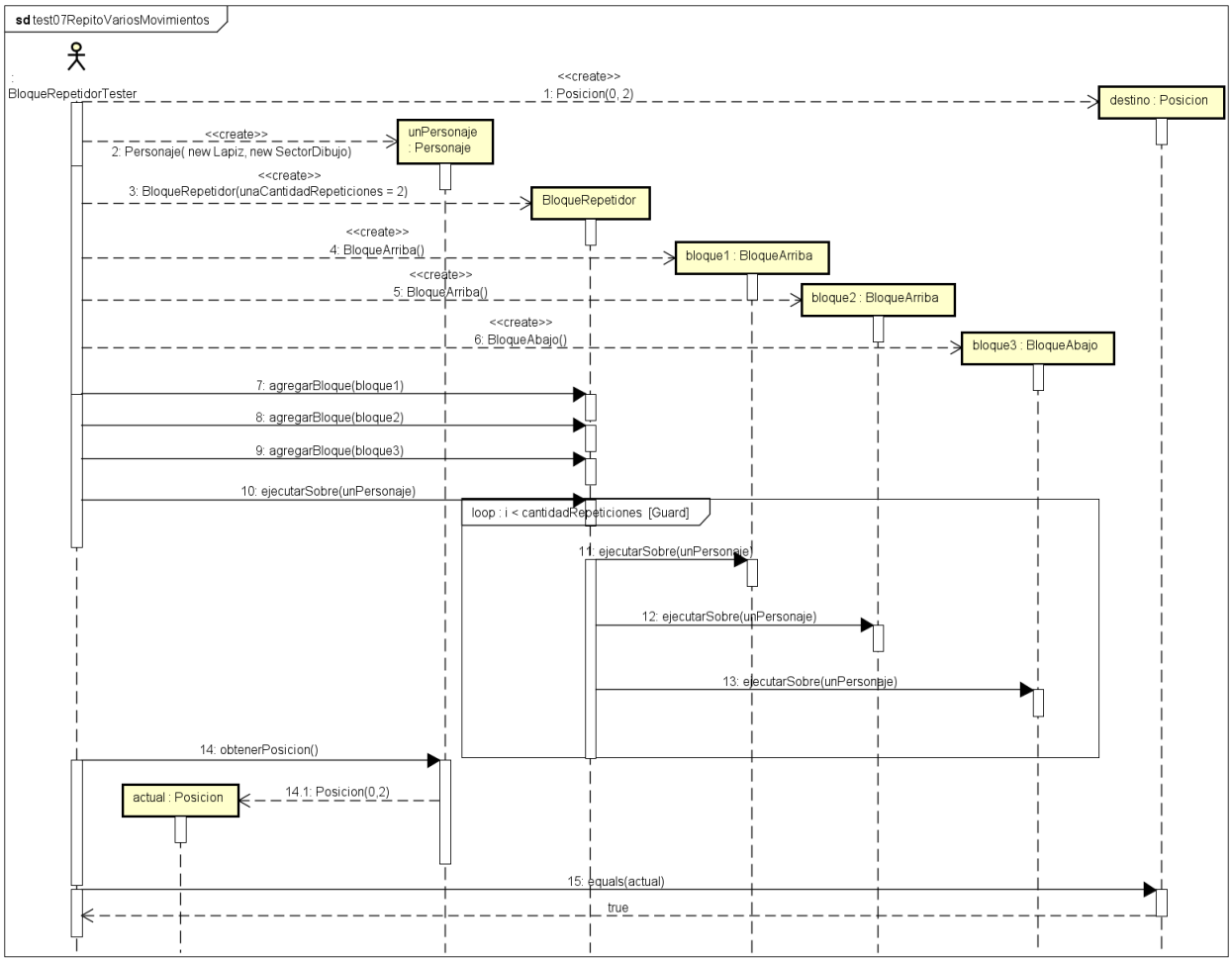

Figura 9: Diagrama sobre la ejecución de un algoritmo con un Bloque de repetición el cuál contiene en su interior varios Bloques de movimiento, los cuales hacen que el Personaje se desplace en varias direcciones.

5.2. Diagramas de implementaciones

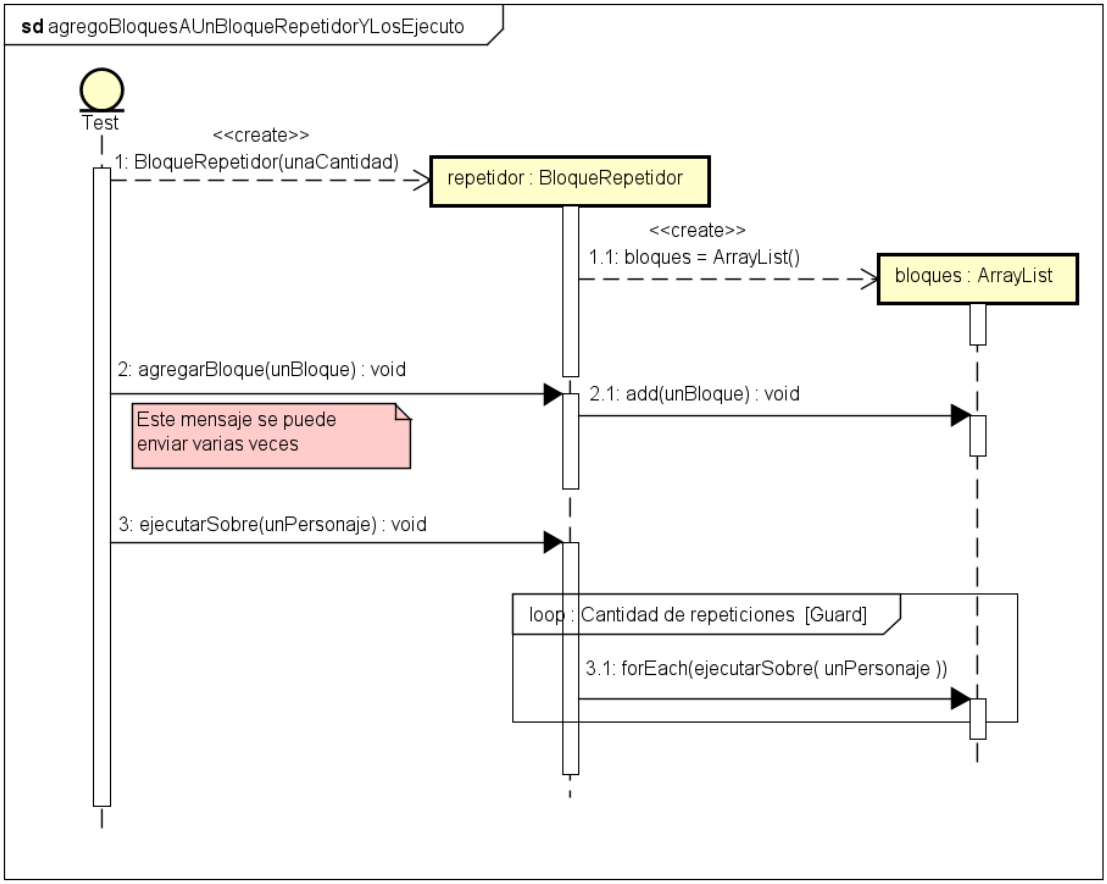

Figura 10: Diagrama de Secuencia de la ejecución de un Bloque Repetidor.

6. Diagrama de Estado

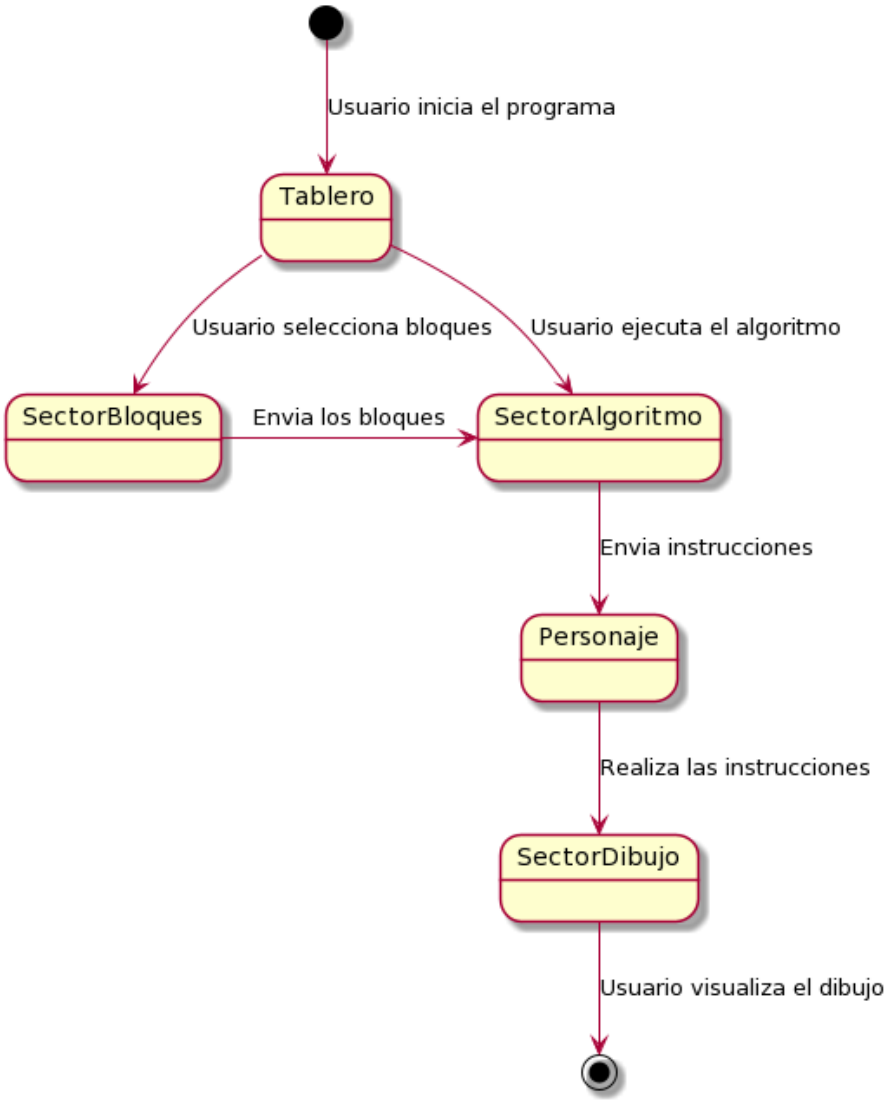

Figura 11: Diagrama de Estado de la ejecución principal del programa.

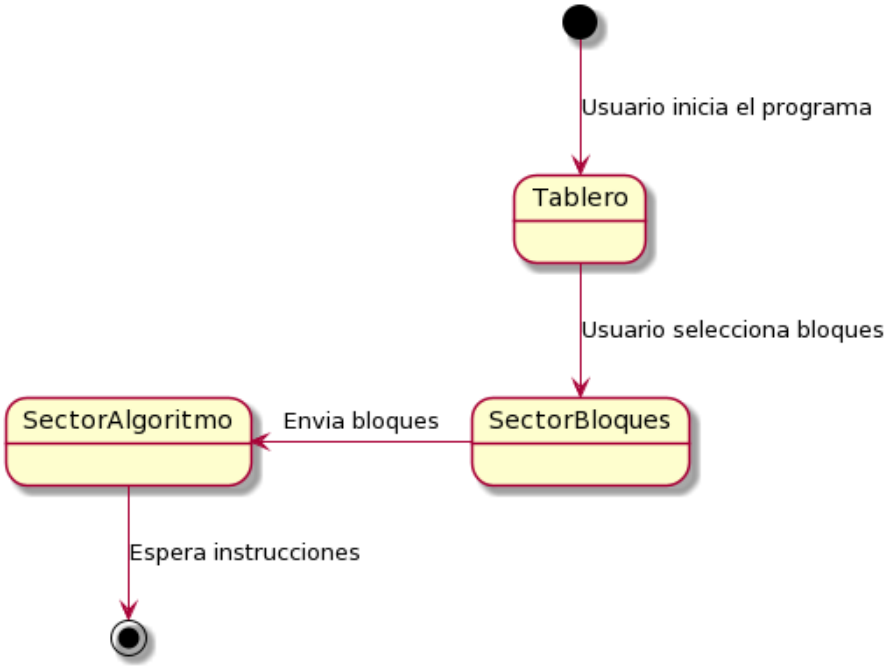

Figura 12: Diagrama de Estado que refleja la selección de bloques, previo a utilizarse un bloque de Algoritmo Personalizado.

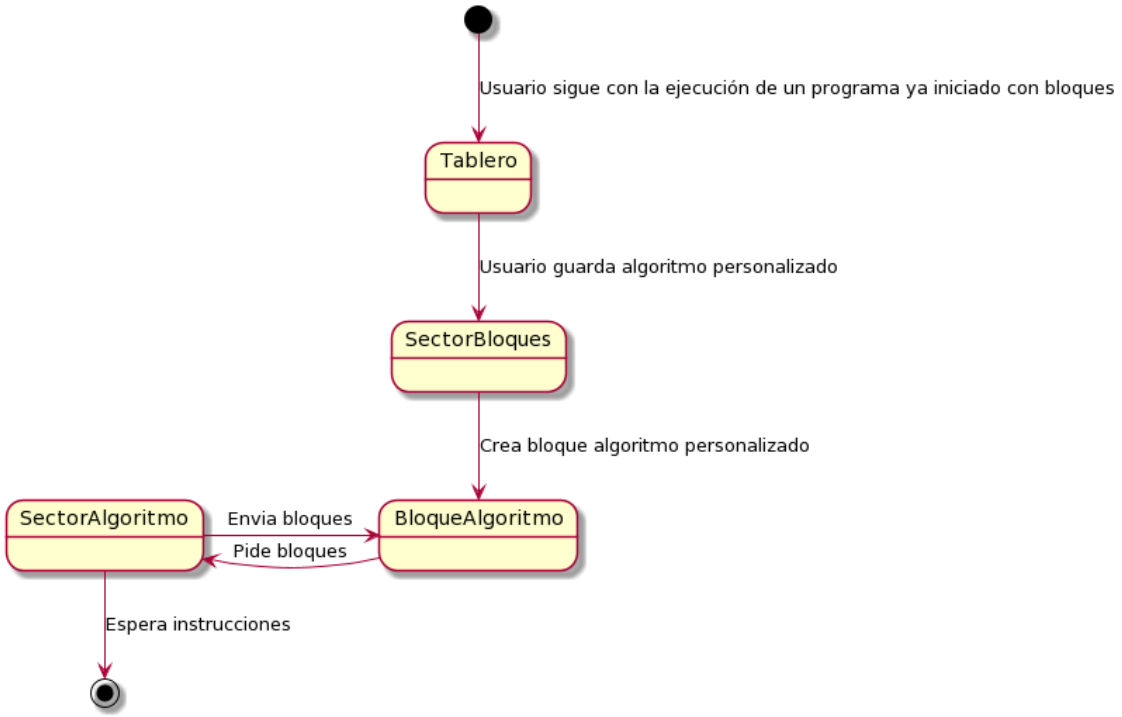

Figura 13: Diagrama de Estado continuado del anterior, que refleja el uso de un bloque de Algoritmo Personalizado.

7. Diagrama de Paquetes

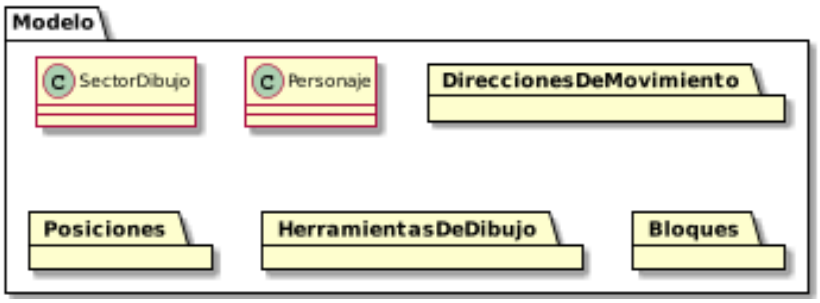

Figura 14: Vista del Paquete Modelo.

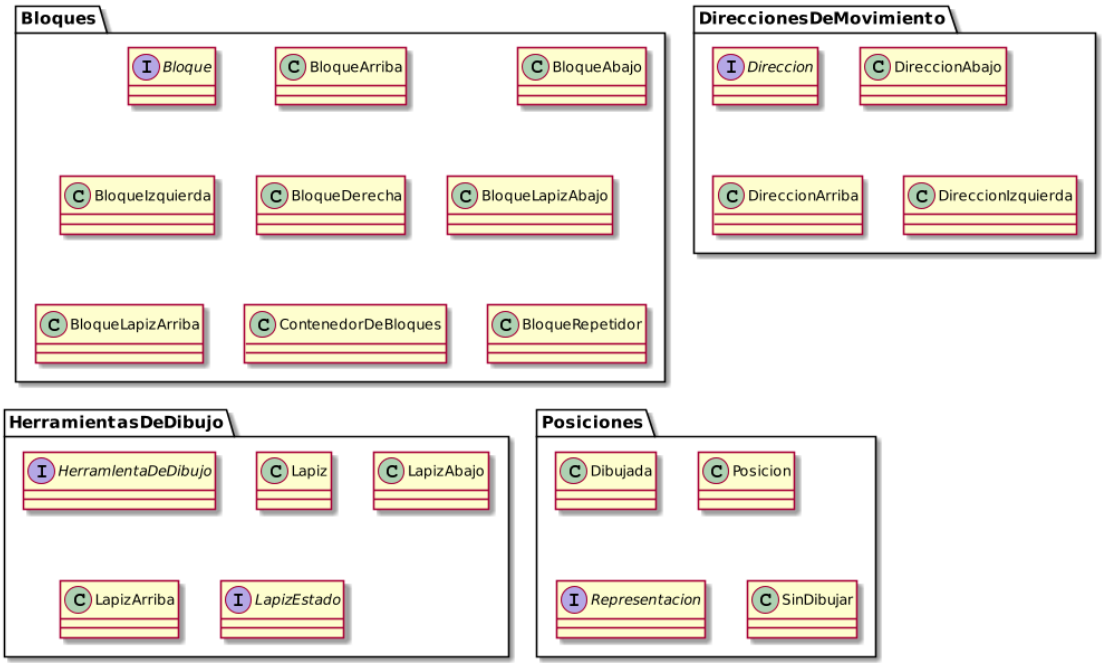

Figura 15: Vista de los paquetes que componen el paquete Modelo.

8. Detalles de Implementación

8.1. Principios de diseño

8.1.1. Principio de Responsabilidad Única

Verificamos que todas las clases que implementamos a lo largo del trabajo cumplan con este principio, manteniendo cada una únicamente una responsabilidad, logrando que el programa sea mucho más fácil de mantener ante posibles cambios.

8.1.2. Principio Abierto-Cerrado

Hacemos uso de este principio puntualmente en el caso del Personaje, el cuál utiliza la interfaz HerramientaDeDibujo, misma que es implementada por la clase Lápiz. Decidimos realizarlo de

esta manera, ya que en un futuro estas relaciones podrían variar ya que quizás el Personaje podría utilizar otra herramienta para dibujar sobre el SectorDibujo. Es así que garantizamos de esta forma que nuestras clases quedan cerradas para modificaciones, pero abiertas al mismo tiempo ante la necesidad de implementar en un futuro nuevas herramientas para dibujar.

8.2. Patrones de diseño

8.2.1. Patrón Strategy, Lápiz, LapizEstado, LapizArriba y LapizAbajo

Implementamos la clase Lápiz, en composición con la interfaz *LapizEstado*, la cual sirve de utilidad para que la herramienta de dibujo Lápiz, internamente pueda adquirir diferentes comportamientos. Para esto, la interfaz posee la firma del método utilizarSobre el cuál las clases LapizAbajo y LapizArriba implementan, interactuando cada una de diferente forma con una Posición en especifico del SectorDibujo. Para llevar adelante esta idea, nos basamos en hacer uso del patrón Strategy, el cuál busca tener un comportamiento que sea solucionado en tiempo de ejecución, ya que a lo largo de la vida útil del personaje este deberá subir y/o bajar su lápiz según se le indique. Estas indicaciones serán llevadas a cabo por la ejecución de los bloques BloqueLapizArriba y BloqueLapizAbajo.

8.3. Diseño en profundidad

8.3.1. Bloques

Para el armado y el uso de Bloques, decidimos implementar una interfaz que lleva el mismo nombre. La cuál es implementada por diferentes tipos de bloques, los cuales varían según su funcionalidad. Estos se pueden utilizar mediante un método el cuál debe recibir al Personaje como parámetro.

8.3.2. Bloques del Lápiz y el Personaje

Para que el Personaje pueda hacer uso a futuro del Lápiz, es condición necesaria que este lo reciba por parámetro en su constructor de clase. Por otro lado, para que el Personaje pueda subir y bajar el Lápiz, es decir para que pueda cambiar su orientación, se deberá recurrir a la ejecución de los bloques BloqueLapizArriba y BloqueLapizAbajo. Estos bloques se comunican con el personaje mediante métodos, los cuales le indican como hacerlo. De esta forma, el Lápiz cambiará su estado internamente con instancias correspondientes a las clases LapizAbajo y LapizArriba. Estas últimas, poseen cada una un comportamiento especifico el cuál se relaciona con el SectorDibujo, al momento de dibujar sobre él. Sin embargo, para que se pueda hacer uso del lápiz, es decir hacer uso de la capacidad de dibujar, se le debe indicar al Personaje que debe moverse con los bloques que poseen esta funcionalidad.

8.3.3. Bloques de movimiento y el Personaje

El movimiento del Personaje está dado por la ejecución de cualquiera de los siguientes bloques, BloqueArriba, BloqueAbajo, BloqueDerecha y BloqueIzquierda, estos le envían al Personaje la dirección de movimiento en la cuál debe moverse, la cuál entiende mediante el uso de un método. El personaje, al recibir la dirección a la cual debe moverse, dibujará sobre la posición en la que se encuentra, luego le enviará un mensaje al SectorDibujo para que cambie su posición y finalmente volverá a dibujar sobre la nueva posición a la que se desplazó.

8.3.4. Bloque de repetición y su relación con otros bloques

Para lograr que el BloqueRepetidor reciba, almacene y ejecute una serie de Bloques se utiliza la estructura de datos ArrayList. La misma se comporta como un Array dinámico y nos permite agregar bloques indefinidamente.

Al ejecutar el BloqueRepetidor se deben ejecutar todos los bloques guardados, en el orden en el que fueron dados, una cantidad de veces, para lograrlo utilizamos el método forEach del Arraylist dentro de un ciclo definido **for**.

```
for (int iteracion = 0; iteracion < cantidadDeRepeticiones; ++iteracion)
    bloques.forEach(ejecutar);
```

En este caso el elemento ejecutar representa un objeto de la clase Consumer, necesario para el método forEach, el cual definimos de la siguiente forma:

```
Consumer<Bloque> ejecutar = bloque -> bloque.ejecutarSobre(personaje);
```

8.3.5. SectorDibujo

Para el SectorDibujo elegimos utilizar una matriz compuesta de Posiciones (inicialmente creadas sin dibujar), la cuál lleva registro de la Posición en la que el Personaje se encuentra. A su vez, entiende como consultar y actualizar la Posición del mismo cuando este necesita movilizarse.

8.3.6. Posición

Llamamos así a los lugares en los que el Personaje puede ubicarse, siendo estos mismos capaces de tomar distintas representaciones, ya sea por estar dibujados o no dibujados. Estarán encargados de comunicarse visualmente con el Usuario por medio del SectorDibujo.

9. Excepciones