FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2

[7507/9502] Algoritmos y Programación III
Curso 1
Segundo cuatrimestre de 2020

COLLAZO, Manuel	100067
DE LUCIA, Lautaro	100203
ESPERON, Ramiro	103992
PUGLISI, Agustín	104245

Índice

1. Introducción **2**

2. Supuestos **2**

3. Diagramas de clase **3**

4. Diagramas de secuencia **4**

5. Diagrama de paquetes **5**

6. Diagrama de estado **6**

7. Detalles de implementación **7**

8. Excepciones **10**

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de manera grupal aplicando todos los conceptos vistos en el curso, utilizando un lenguaje de tipado estático (Java) con un diseño del modelo orientado a objetos y trabajando con las técnicas de TDD e Integración Continua.

2. Supuestos

Supuesto 1: Todos los elementos son únicos. Esto es, solo hay un tablero, y consecuentemente (dado que están contenidos en él), solo un personaje y un lápiz. También, si bien hay muchos bloques, e incluso bloques de bloques, la secuencia a ejecutar es única y se construye acoplando bloques y sub-secuencias de bloques. Esta secuencia tiene que estar completamente definida previo a toda ejecución.

Supuesto 2: Si el personaje alcanza los límites del tablero, no se ejecutará ninguna acción por fuera de este (la secuencia no se dibujará más allá de los límites del tablero).

Supuesto 3: El bloque personalizado es uno y sólo uno (agregar múltiples bloques personalizados complicaría la vista). Si se quiere agregar una nueva secuencia personalizada, debe borrarse la anterior.

Supuesto 4: Luego de ejecutar una secuencia de bloques que muevan y alteren el estado del personaje, esta secuencia no se borrará hasta que lo indique el usuario. De esta manera es posible armar una secuencia de bloques, ejecutarla, y luego seguir agregando bloques a la misma, obteniendo una secuencia mas larga que podrá ser ejecutada nuevamente.

Supuesto 5: A las funcionalidades explicitadas en la consigna, se agregan i) Un borrador para el lápiz y ii) la capacidad de que el lápiz tenga distintos colores. Estos colores se configuran durante la secuencia, esto es, se manifiestan mediante el agregado de "Bloques de Color". Al mismo tiempo, la capacidad de levantar el lápiz, bajarlo, y ponerlo en borrador son también acciones que se ejecutan como bloques dentro de la secuencia.

3. Diagramas de clase

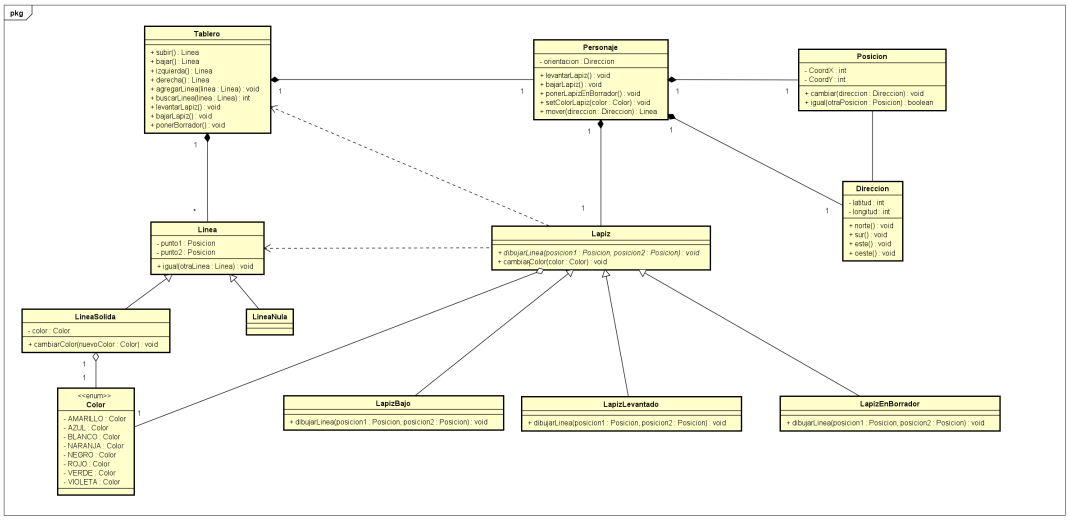

Figura 1: Sector dibujo.

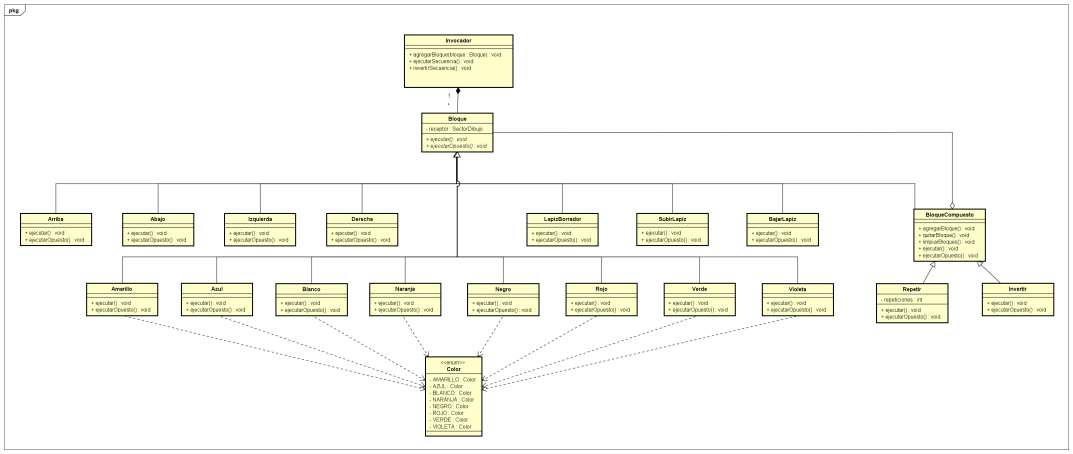

Figura 2: Sector bloques.

4. Diagramas de secuencia

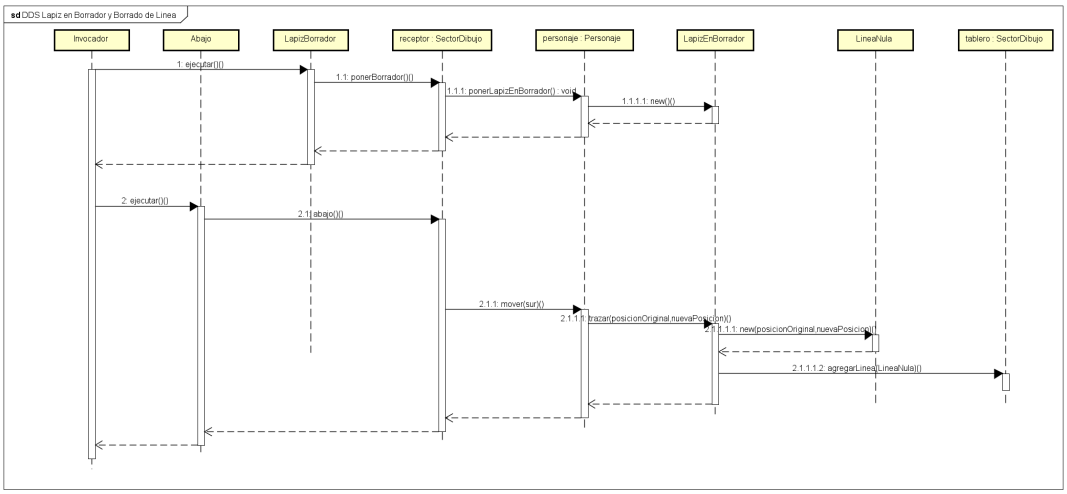

Figura 3: Lápiz en borrador y borrado en linea.

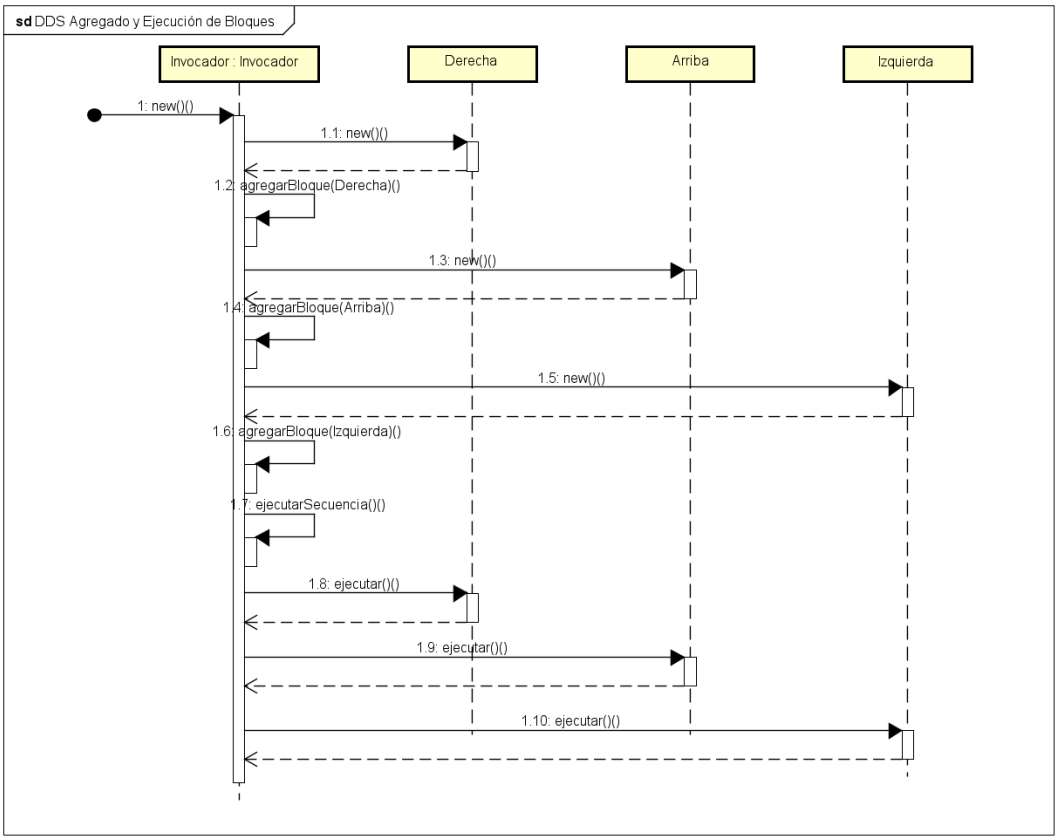

Figura 4: Agregado y ejecución de bloques.

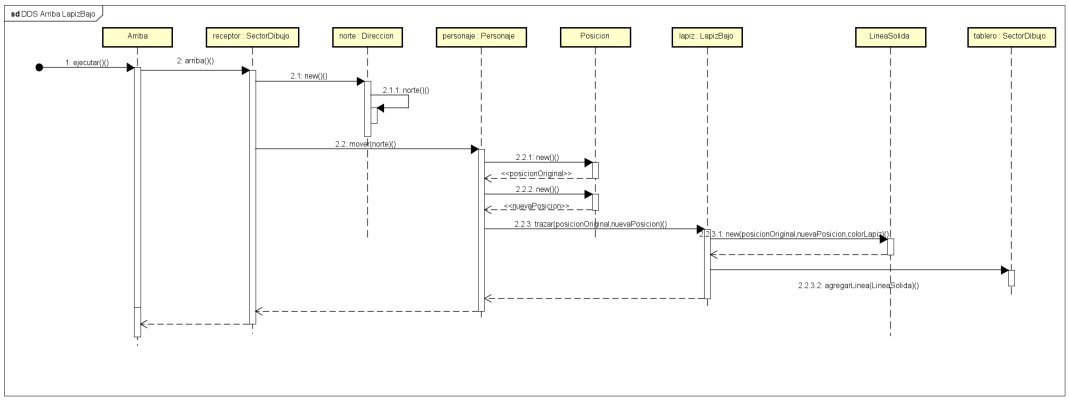

Figura 5: Arriba LapizBajo.

5. Diagrama de paquetes

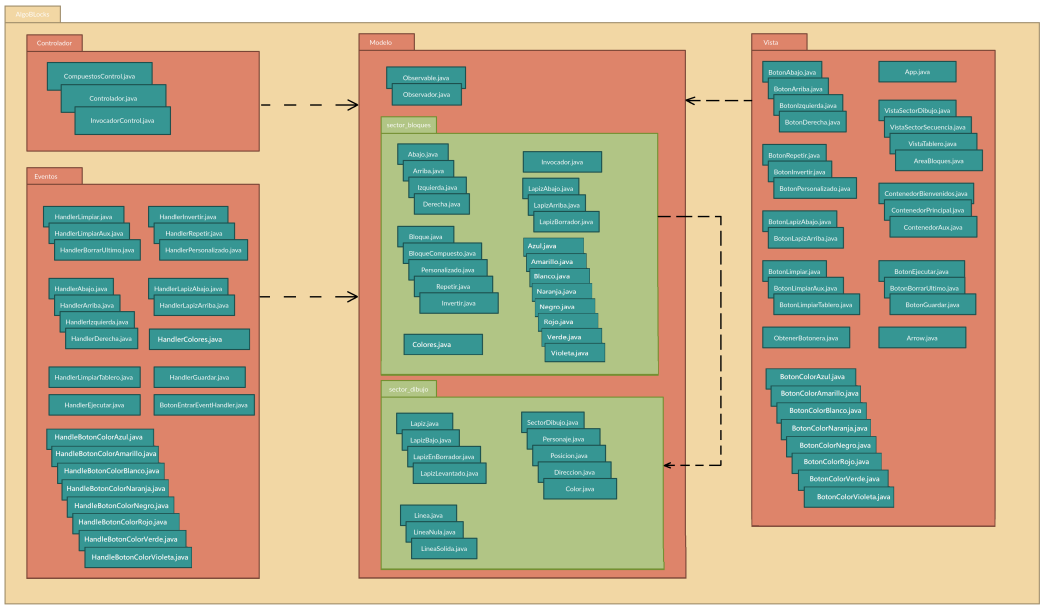

Figura 6: Diagrama de paquetes.

6. Diagrama de estado

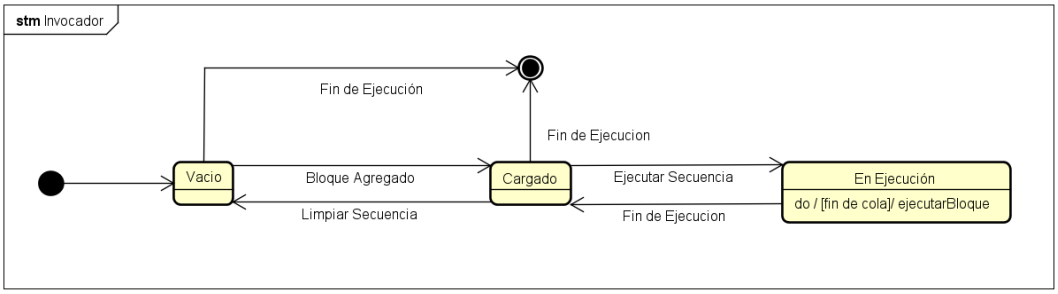

Figura 7: Invocador.

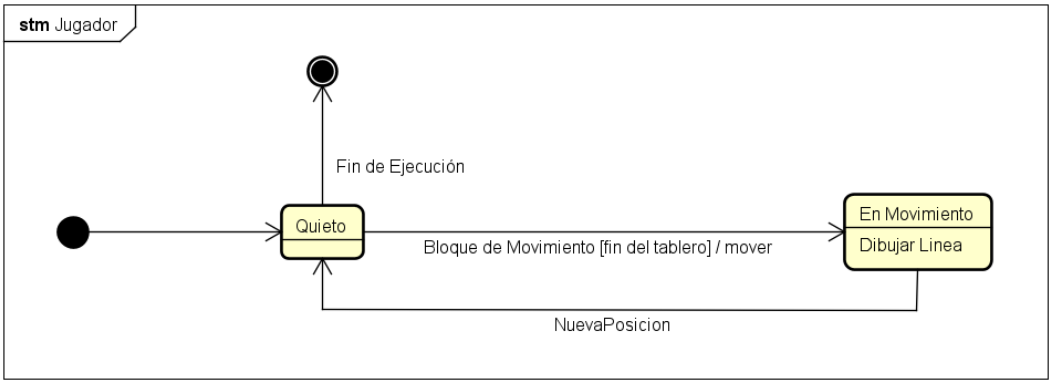

Figura 8: Jugador.

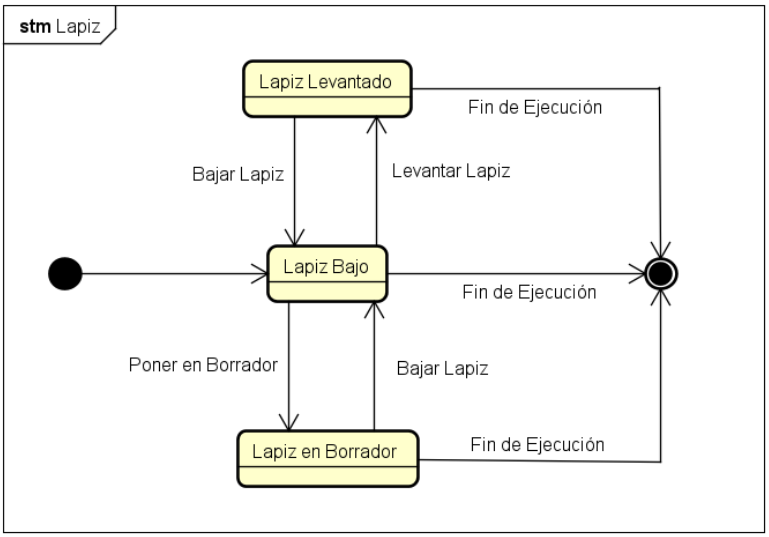

Figura 9: Lapiz.

7. Detalles de implementación

Debido a la naturaleza del programa, en la cual **primero** el usuario debe definir una secuencia de acciones y **después** estas deben ejecutarse para formar un dibujo en un tablero, se decidió dividir el programa en dos módulos principales: **Sector Bloques** y **Sector Dibujo**. El primero está encargado de todo lo referente a la secuencia de acciones mientras que el segundo se encarga de todo lo referente al dibujo.
Para la implementación de los bloques comunes se optó por implementar el PDD **Command**. De esta forma, cada acción específica se encapsula dentro de una clase heredera de la clase abstracta principal ***Bloque***. Esta es una clase que tiene sólo un atributo: el *receptor* de la acción, que en nuestro caso será una instancia de la clase ***SectorDibujo***.
A modo de evitar posibles errores, se implementó el patrón **Singleton** para la clase ***SectorDibujo***, de modo que haya una y solo una instancia de la clase en todo el programa (lo que es coherente, ya que solo hay un solo dibujo); de este modo, se evita que, por ejemplo, dos bloques dentro de la secuencia estén asignados a receptores distintos.
La clase abstracta ***Bloque*** tiene dos métodos abstractos **ejecutar()** y **ejecutarOpuesto()**, los cuales se redefinen en cada heredero de la clase ***Bloque***, donde cada uno representa una acción específica a ejecutarse sobre el dibujo. Entre estas están las de movimiento (***Arriba***, ***Abajo***, etc...), las que afectan al estado del lápiz (***SubirLapiz***, ***BajarLapiz***, ***LapizEnBorrador***), así como las que definen el color del dibujo (***Amarillo***, ***Verde***, etc...).

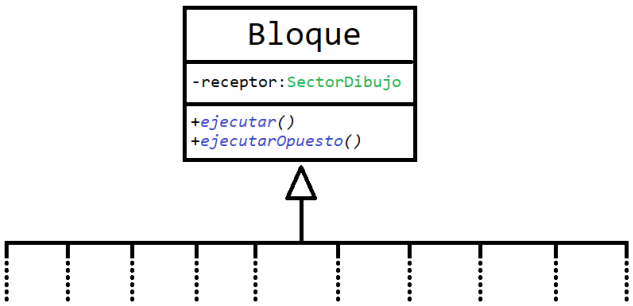

Figura 10: Estructura de la clase bloque.

Para la implementación de los bloques *compuestos*, se opto por utilizar el patrón **Composite**. De esta forma, un bloque compuesto sigue siendo tratado como un bloque común para el programa. Dentro del bloque compuesto redefinimos los métodos **ejecutar()** y **ejecutarOpuesto()** de modo que recorran la lista de bloques que se agregaron dentro del bloque compuesto, ejecutando estos mismos métodos en cada uno. Cada bloque individual contenido en el compuesto conoce su propia implementación y no se requiere más que esto para definir un *bloque de bloques*.

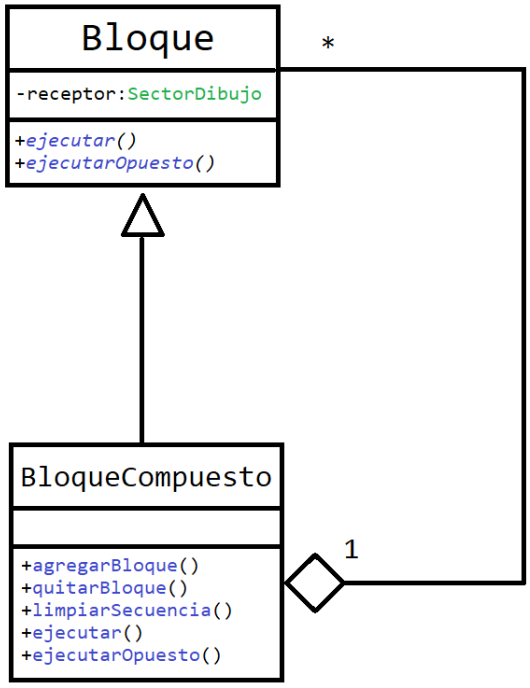

Figura 11: Bloque compuesto.

Luego, los bloques ***Personalizado***, ***Invertir*** y ***Repetir*** constituyen sub-clases de este bloque compuesto, cada una con su implementación particular explícita en la redefinición de los métodos **ejecutar ()** y **ejecutarOpuesto()**.
Todos estos posibles bloques se ponen en la secuencia tal como es definida por el usuario. Esta secuencia esta representada programáticamente por la clase ***Invocador*** la cual contiene el arreglo principal con todos los bloques. Cuando el usuario ejecuta la secuencia, lo único que ocurre es que el arreglo de bloques dentro de ***Invocador*** se recorre, dando a **ejecutar()** en cada uno.

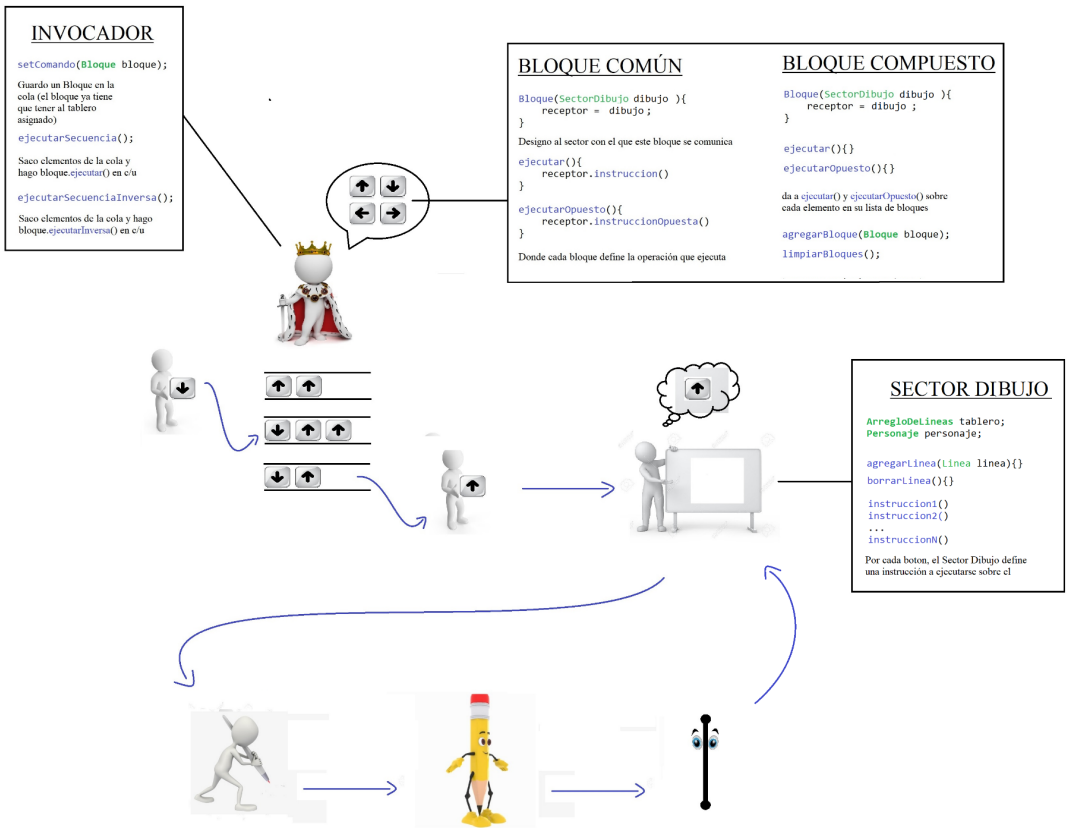

Figura 12: Lógica del invocador.

Estos bloques comunican la acción a realizarse a su receptor, esto es, la instancia única de la clase ***SectorDibujo***, la cual interactúa con el ***Personaje*** y el ***Lapiz*** para llevarlas a cabo. Estos están definidos como atributos dentro de la instancia de la clase ***SectorDibujo***, por lo que son a su vez únicos.

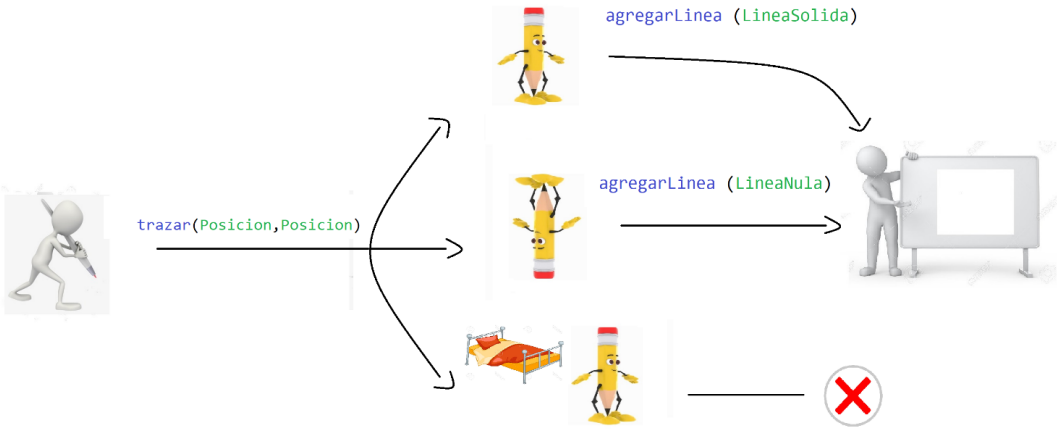

Figura 13: Comportamiento de la clase Lapiz.

Cuando el personaje recibe la instrucción de moverse de parte del dibujo, le comunica al lápiz que debe trazar una nueva línea entre los puntos P1 y P2, siendo cada uno respectivamente la

posición original y la nueva posición del jugador. La forma en la que la línea se traza depende del estado del lápiz. Si este esta bajo, traza una ***LineaSolida*** del color del lápiz; si está en borrador, traza una ***LineaNula***; si esta levantado, no realiza ninguna acción. El lápiz se encarga de comunicarse con el tablero. Por ejemplo, si se traza una línea sólida de color rojo, es responsabilidad de el lápiz i) crearla y ii) agregarla a el dibujo.
Las líneas sólidas son interpretadas por la vista, y son aquellas que serán visualizadas por el usuario. Una ***LineaNula*** no es más que la implementación del patrón **NullObject** sobre la clase ***Linea***. En realidad, representan esencialmente un *espacio vacío* en el dibujo. El tablero de dibujo (atributo dentro de la clase ***SectorDibujo***) es un arreglo de líneas. Las líneas nulas representan espacios vacíos mientras que las líneas sólidas *rellenan* este espacio. Por defecto, un tablero de dibujo que se inicializa vacío esta en realidad lleno de líneas nulas. Luego, si el personaje se mueve de P1 a P2 con el lápiz bajo, se reemplaza en el tablero la línea nula que estaba entre P1 y P2 por una línea sólida del color del lápiz y esta se hará visible en la vista.

8. Excepciones

FueraDeLimiteExcepcion: Esta excepción es arrojada cuando el ***Personaje*** intenta moverse más allá de los limites del tablero (***SectorDibujo***). Ante este escenario, el ***Personaje*** debe permanecer en el lugar y no avanzar a menos que una instruccion posterior indique que vuelva al área permitida del **SectorDibujo**.

CantidadMaximaDeBloquesAlcanzadaExcepcion: Como se decidió implementar el registro de bloques con una **ArrayList**, se decidió fijar una excepción para cuando se llegue a cierta cantidad máxima de bloques permitidos acorde a nuestra aplicación que se ejecute antes de cualquier excepción propia de **ArrayList** referente al tamaño del mismo.