FACULTAD DE INGENIERIA
Universidad de Buenos Aires

# Trabajo Práctico 2 — Java

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


| Alumno | Padron | Mail |
|---|---|---|
| Michael Gustavo Mena Giraldo | 102685 | mgmena@fi.uba.ar |
| Juan Ignacio Karabin | 104697 | jkarabin@fi.uba.ar |
| Juan Manuel Prato | 105224 | jprato@fi.uba.ar |

# Índice

# 1. Introducción

El presente informe reune la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicacion en Java que permita lo siguiente:

- Crear un personaje que pueda levantar o bajar un lápiz.
- Mover al personaje en ciertas direcciones utilizando bloques.
- Crear un sector dibujo.
- Mover al personaje con su lápiz y realizar un dibujo cuando el mismo se encuentra bajo.
- Crear algoritmos utilizando bloques de repetición.
- Crear algoritmos utilizando bloques de invertir.

# 2. Supuestos

- El personaje no puede moverse en diagonal.
- Por defecto, el personaje comienza con el lápiz levantado.
- El tablero es infinito.

# 3. Modelo de dominio

Para comenzar con nuestro modelo de dominio, daremos una lista de las clases mas relevantes que fueron creadas para luego dar mas detalles acerca de estas:

- Contenedor de Bloques
- Sector Dibujo
- Sector Algoritmo
- Lápiz
- IBloque

**ContenedorDeBloques** Se implementó esta clase abstracta ya que varias otras clases necesitan la capacidad de almacenar bloques, e interactuar con estos. Por ejemplo, el sector algoritmo, o los bloques repetir e invertir.

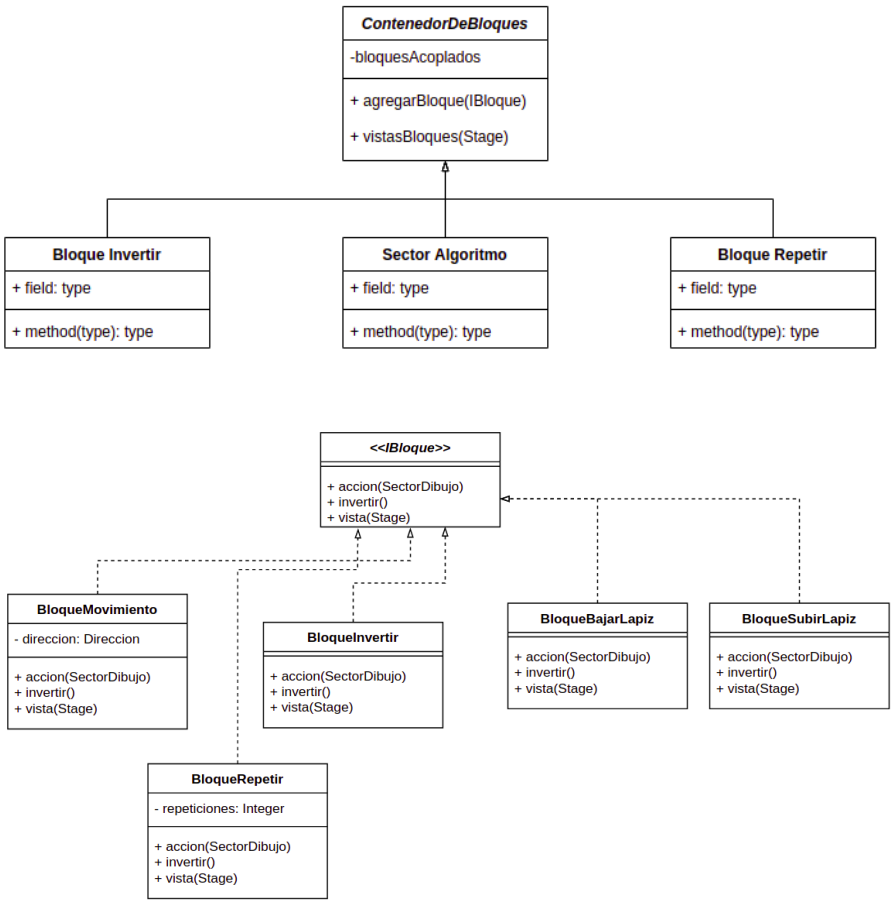
**SectorDibujo** Comenzando con el sector dibujo, este es es el encargado de contener tanto el tablero como el personaje, con esta clase se utilizó el patron Singleton, ya que esta es una clase en la cual solo se una instancia de la misma.

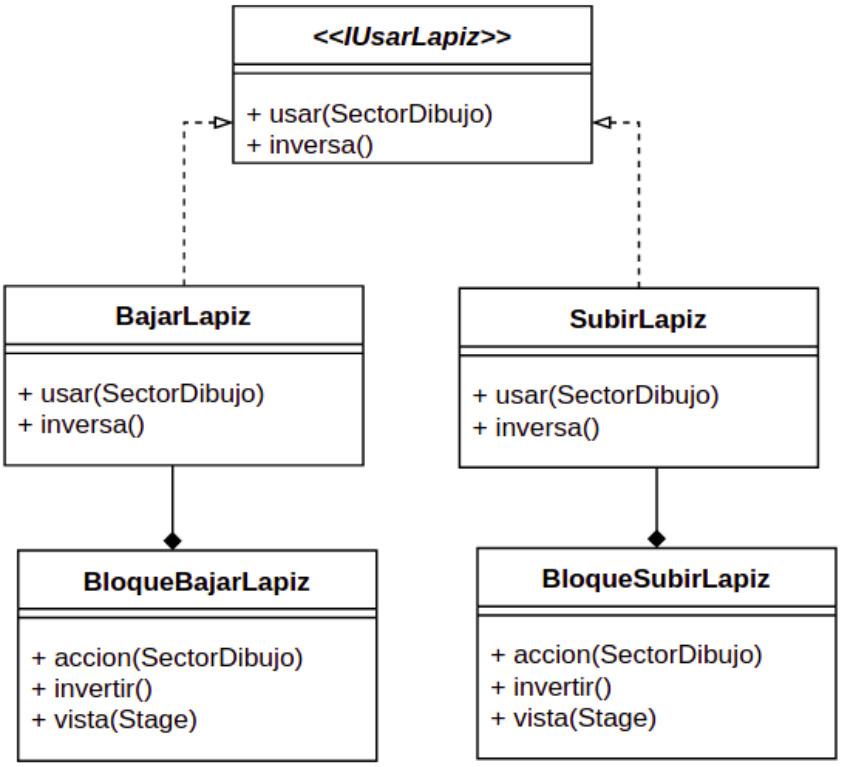
**SectorAlgoritmo** En esta clase implementa el patron Singleton, ya que solo se requiere una instancia del mismo. Ademas de esto, esta clase hereda de ContenedorDeBloques, ya que en esta clase se almacenan los bloques del algoritmo a ejecutar.

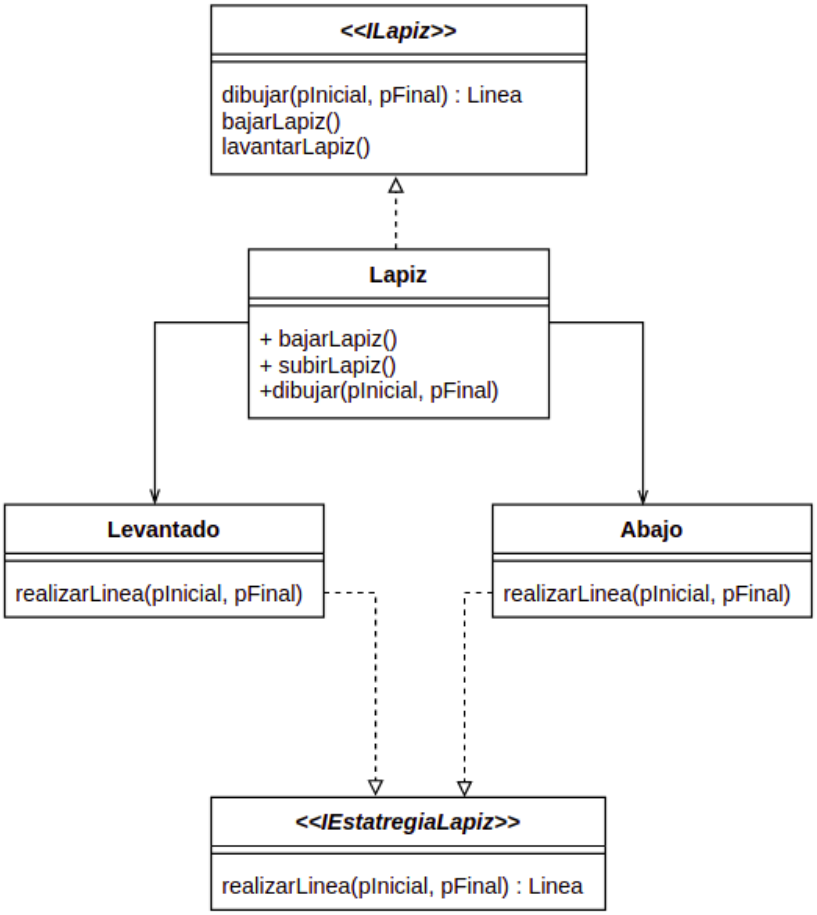
**Lápiz** La instancia de esta clase, puede comportarse de dos formas distintas. Si se encuentra levantado, es decir, con su punta hacia arriba, no dibuja ninguna línea. Caso contrario, cuando su punta se encuentra boca abajo, dibuja las respectivas líneas.

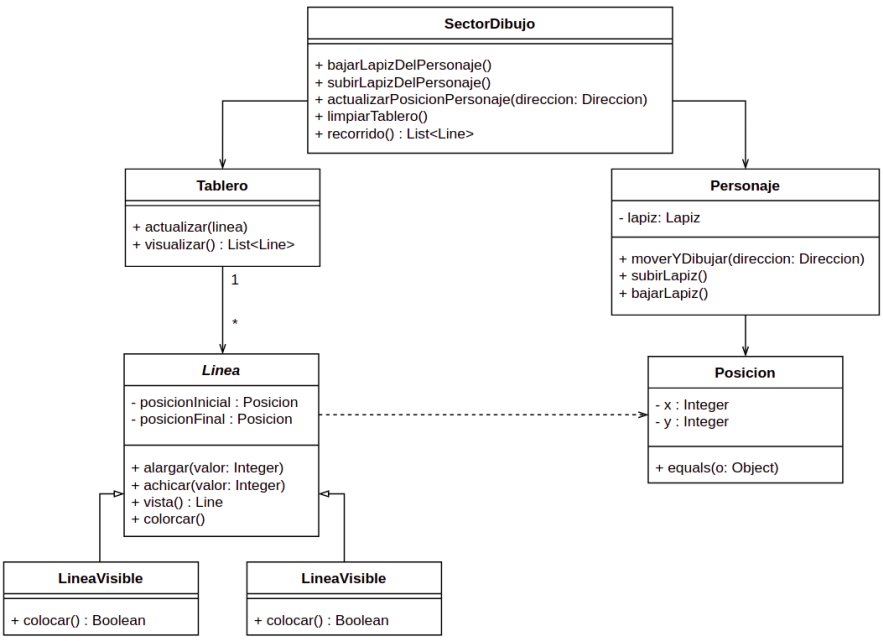
**IBloque** Se crea esta interfaz para que la implemente cada bloque, ya que todos los bloques deberia poder responder a los mensajes declarados en esta.

# 4. Diagramas de clase




<<IUsarLapiz>>
+ usar(SectorDibujo)
+ inversa()
BajarLapiz
+ usar(SectorDibujo)
+ inversa()
SubirLapiz
+ usar(SectorDibujo)
+ inversa()
BloqueBajarLapiz
+ accion(SectorDibujo)
+ invertir()
+ vista(Stage)
BloqueSubirLapiz
+ accion(SectorDibujo)
+ invertir()
+ vista(Stage)


<<ILapiz>>
dibujar(pInicial, pFinal) : Linea
bajarLapiz()
lavantarLapiz()
Lapiz
+ bajarLapiz()
+ subirLapiz()
+dibujar(pInicial, pFinal)
Levantado
realizarLinea(pInicial, pFinal)
Abajo
realizarLinea(pInicial, pFinal)
<<IEstatregiaLapiz>>
realizarLinea(pInicial, pFinal) : Linea




# 5. Detalles de implementación

## 5.1. Clase Lápiz

En el caso de esta clase, se utilizó el patron strategy, ya que el lápiz debía poder comportase como un lápiz levantado, y en tal caso no debe dibujar una linea, y como uno apoyado, en ese caso, debiendo dibujar la línea. Para esto, el lápiz puede usar la estrategia del lápiz lavantando o apayodo. Este cambio de estrategia es realizado externamente, segun los bloques que se utilizen.

# 6. Diagrama de paquetes