A picture containing text

Description automatically generated

Trabajo Práctico 2 — Java

[7507/9502] Algoritmos y Programación III Curso 2 (noche)

Segundo cuatrimestre de 2020

|  |  |
| --- | --- |
| Alumnos: | xxx |
| Número de padrón: | xxx |
| Email: | xxx |

**Índice**

1. [**Introducción**](#_Introducción)
2. [**Supuestos**](#_Supuestos)
3. [**Diagramas de clase**](#_Diagramas_de_clases)
4. [**Diagramas de secuencia**](#_Diagramas_de_secuencia)
5. [**Diagramas de paquetes**](#_Diagramas_de_paquetes)
6. [**Diagramas de estado**](#_Diagramas_de_estado)
7. [**Detalles de implementación**](#_Detalles_de_implementación)
   1. [*Lapiz, Estado Lapiz y patrón State*](#_Lapiz,_Estado_Lapiz)
   2. [*Bloque Mover, Direccion y el patrón Factory*](#_Bloque_Mover,_Dirección)
   3. [*Observadores, Observados y el patrón MVC*](#_Observadores,_observados_y)
8. [**Expeciones**](#_Excepciones)

# Introducción

# Supuestos

En el presente trabajo, se realizaron los siguientes supuestos:

* Ningún tipo de secuencia de bloques, ya sea bloque repetir, bloque invertir, bloque personalizado, o el algoritmo a ejecutar, puede ejecutarse y/o guardarse vacío. Debe contener al menos un bloque en su interior.
* Se detalla la operación *invertir* de cada tipo de bloque

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de bloque** | **Bloque Ingresado** | **Acción invertida** | **Ejemplo** |
| **Bloque Simple** | Mover Derecha | mueve a izquierda |  |
|  | Mover Izquierda | mueve a derecha |  |
|  | Mover Arriba | mueve abajo |  |
|  | Mover Abajo | mueve arriba |  |
| **Bloques Complejos** | Algoritmo[[1]](#footnote-1) | invierte cada bloque | Si ingresan: Mover Arriba, Mover Derecha, Subir Lapiz, se ejecutará: Mover Abajo, Mover Izquierda, Bajar Lapiz |
|  | Repetición | invierte el orden de los bloques | Si ingresan: Repetirx2 (Mover Arriba, Mover Derecha, Subir Lapiz), se ejecutará: Repetirx2(Subir Lapiz, Mover Derecha, Mover Arriba) |
|  | Invertir | inverte el orden de los bloques | Si ingresan: Invertir{ Invertir ( Mover Arriba, Mover Derecha, Subir Lapiz ) }, se ejecutará: Bajar Lapiz, Mover Izquierda, Mover Abajo |
|  | Algoritmo Personalizado | invierte el orden de los bloques | Si el bloque personalizado es: Mover Arriba, Mover Derecha, Subir Lapiz, se ejecutará: Subir Lapiz, Mover Derecha, Mover Arriba |

# Diagramas de clases

Varios diagramas de clases, mostrando la relación estática entre las clases. Pueden agregar todo el texto necesario para aclarar y explicar su diseño de manera tal que logre el modelo logre comunicarse de manera efectiva.

# Diagramas de secuencia

Varios diagramas de secuencia, mostrando la relación dinámica entre distintos objetos planteando una gran cantidad de escenarios que contemplen las secuencias más interesantes del modelo.

# Diagramas de paquetes

Incluir un diagrama de paquetes UML para mostrar el acoplamiento de su trabajo.

# Diagramas de estado

Incluir diagramas de estados, mostrando tanto los estados como las distintas transiciones para varias entidades del modelo.

# Detalles de implementación

Deben detallar/explicar qué estrategias utilizaron para resolver todos los puntos más conflictivos del trabajo práctico. Mencionar qué patrones de diseño fueron utilizados y por qué motivos.

### Lapiz, Estado Lapiz y el patrón State

El objeto Lapiz tiene dos estados posibles:

* estar levantado, que implica que el personaje, al moverse, no pintará;
* estar apoyado, que le permite al usuario dibujar en la pizarra.

Aunque al principio intentamos implementar dos tipos de lápices distintos y que se fuesen pisando el uno al otro cuando el usuario lo necesitaba, pronto nos dimos cuenta de que el patrón State resolvía el comportamiento deseado de una manera mucho más elegante.

De esta manera, el lápiz siempre es el mismo, pero lo que cambia es su estado, que tendrá su comportamiento particular. El usuario, al interactuar con el programa, podrá cambiar el estado del lápiz y a través del polimorfismo y herencia propios del patrón, todos los estados entenderán el mensaje pintar().

El Lapiz, a su vez, posee como atributo un dibujo, que es el que modificará (o no) según el estado correspondiente. De esta manera, el lápiz le delega a su estado la acción de pintar, cosa que el usuario, desde afuera interactuando con la aplicación, no necesita saber.

### Bloque Mover, Dirección y el patrón Factory

### Observadores, observados y el patrón MVC

# Excepciones

Las excepciones creadas para este trabajo fueron las siguientes:

**AlgoritmoVacioExcepcion** Se lanza cuando el usuario intenta ejecutar una secuencia de bloques vacía y cuando se desea guardar un bloque personalizado, también vacío.

**BloquePersonalizadoNoExisteExcepcion** Se lanza cuando el usuario quiere buscar un bloque personalizado con un nombre que nunca se registró. Nota: esta excepción no puede lanzarse desde la interfaz gráfica, ya que el usuario no ingresa e nombre del algoritmo que quiere. Sin embargo, desde el programa es posible mandar un nombre incorrecto al método que busca los bloques personalizados, por lo que la excepción debió tenerse en cuenta.

**BloquePersonalizadoYaExisteExcepcion** Se lanza cuando el usuario quiere agregar un nuevo bloque personalizado con un nombre que ya le pertenece a otro bloque. En ese caso, deberá ingresarse otro nombre.

1. Algoritmo hace referencia a la suciesión de bloques que se encuentra en la interfaz gráfica, la secuencia principal que se ejecuta cuando el usuario agrega bloques y presiona Ejecutar. [↑](#footnote-ref-1)