**Arquitectura del sistema**

**Introducción**

**Patrón arquitectónico:** Un patrón arquitectónico se puede considerar como una descripción abstracta estilizada de buena práctica, que se ensayó y puso a prueba en diferentes sistemas y entornos. De este modo, un patrón arquitectónico debe describir una organización de sistema que ha tenido éxito en sistemas previos.

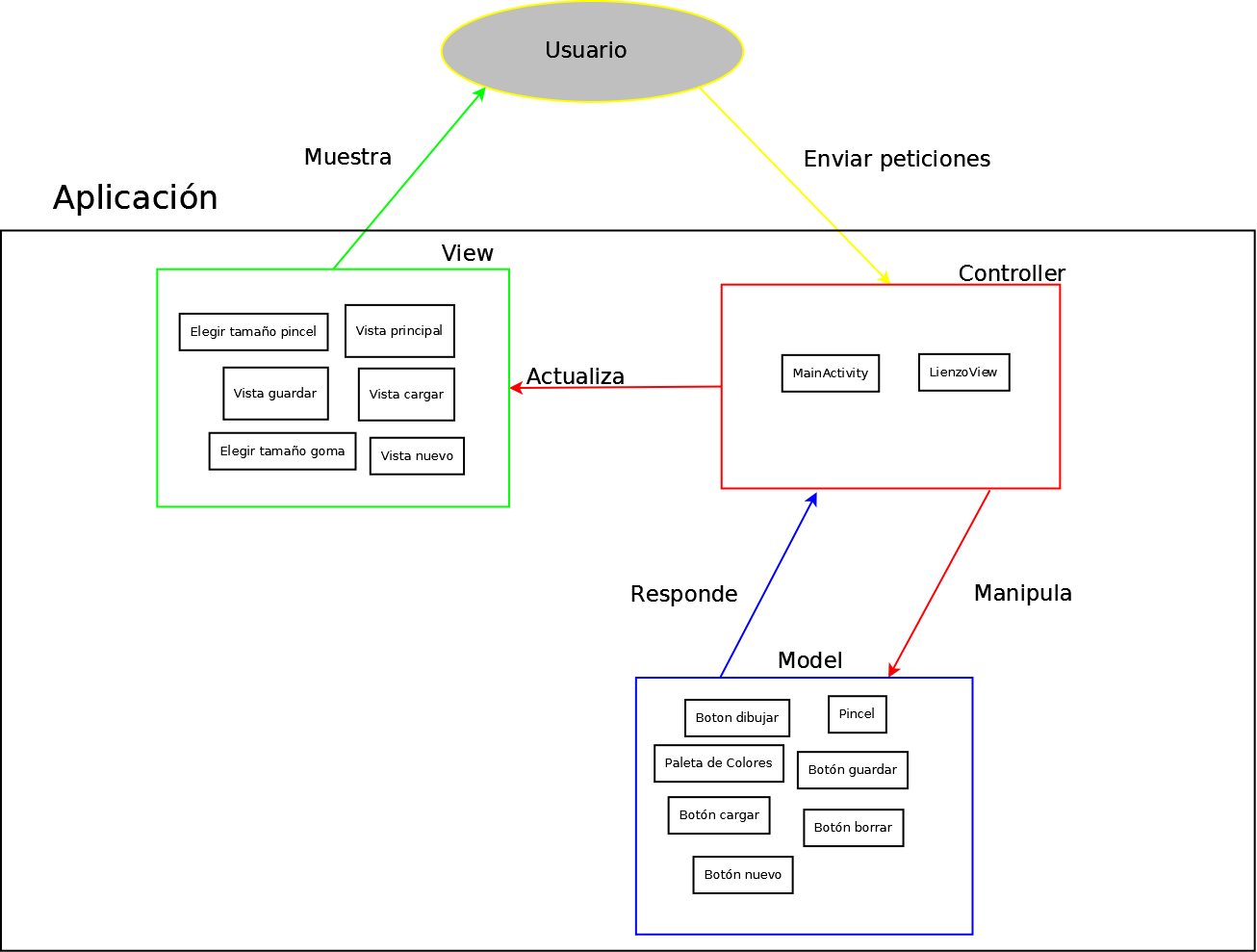
**MVC (Model view controller)**

**Descripción:** Separa la presentación e interacción de los datos del sistema. El sistema se estructura en tres componentes lógicos que interactúan entre sí. El componente ***Model*** (Modelo) maneja los datos del sistema y las operaciones asociadas a esos datos. El componente ***View*** (Vista) define y gestiona como se presentan los datos al usuario. El componente ***Controller*** (Controlador) dirige la interacción de usuario (por ej.: teclas oprimidas, clics del mouse, etc.) y pasa estas interacciones a ***View*** y ***Model***.

**Arquitectura del proyecto**

En nuestro caso el patrón de arquitectura MVC nos es muy útil, ya que al ser una aplicación con la cual el usuario interactúa a través de la pantalla del teléfono, haciendo clic sobre la misma, separar los componentes del sistema en View, Model y Controller nos permite mostrar una View bien definida al usuario ó interfaz de usuario, el cual podrá interactuar con la aplicación a través del Controller que recibirá las acciones que este realice y notificará al Model para así manejar los datos.

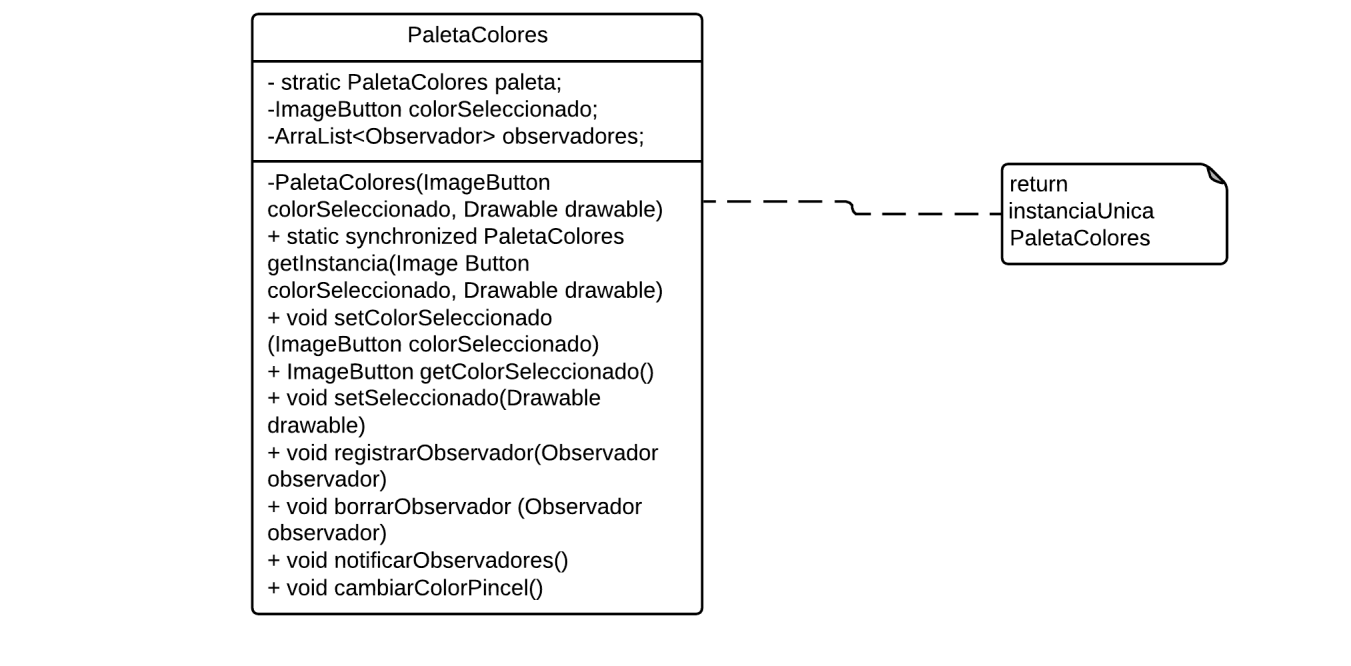
Así las Vistas (View) quedan separadas de la lógica del sistema (Model) permitiéndonos que los datos cambien de manera independiente de su presentación y viceversa. Además es posible construir nuevas Vistas (View) sin necesidad de modificar el Modelo (Model) subyacente.

**Diagrama del patron de arquitectura MVC de la aplicación**

**Diseño del Sistema**

**Patrón Singleton**

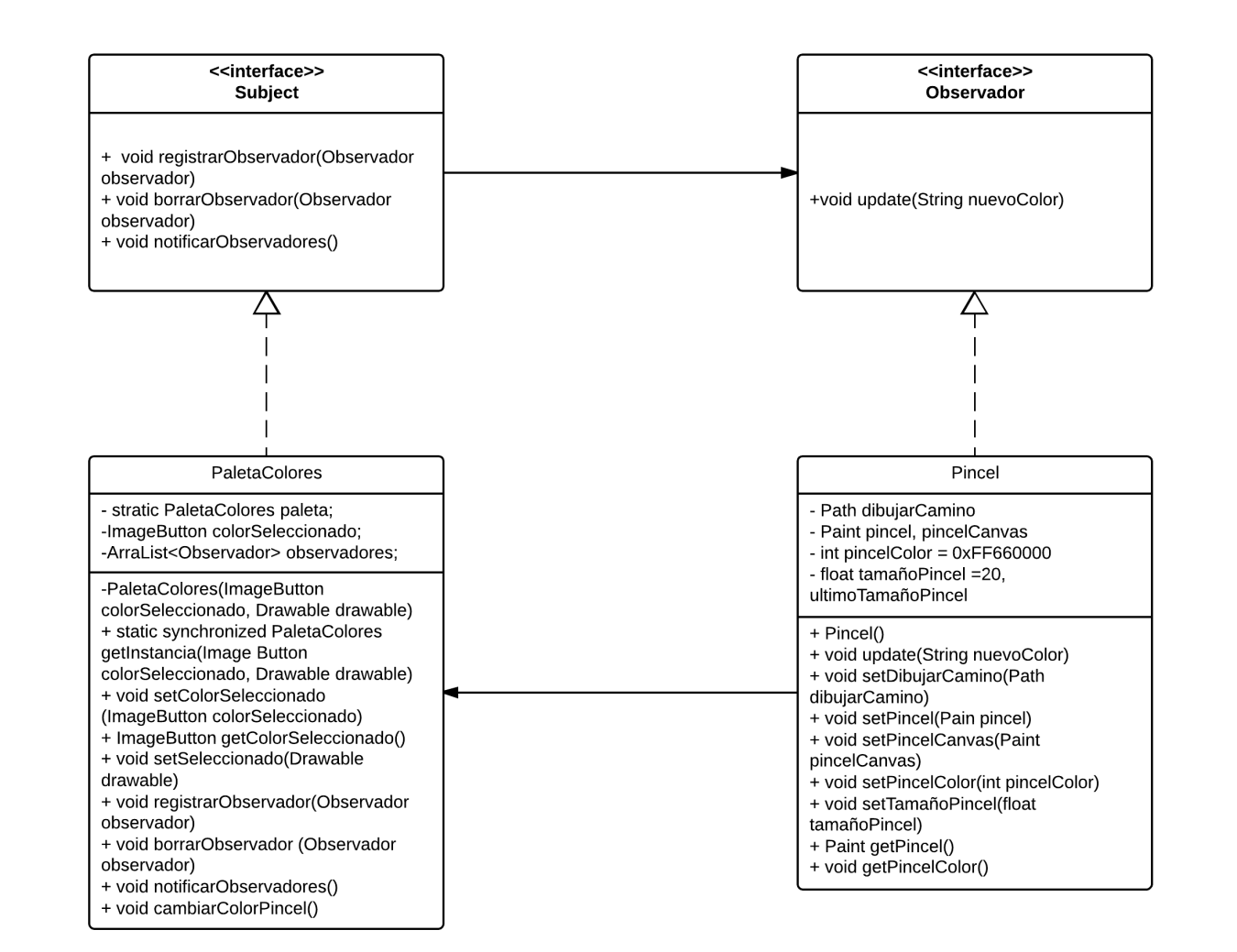
Como el patrón singleton, lo que hace es limitar la creación de una cierta clase a un solo objeto, entonces, pensamos que este patrón debe ser aplicado en la clase Paleta de Colores, ya que en el sistema sólo podrá crearse un objeto Paleta de Colores distinto por cada edición que realice. Por lo cual, tenemos en cuenta esto para realizar la elección del patrón al objeto de la aplicación.

****

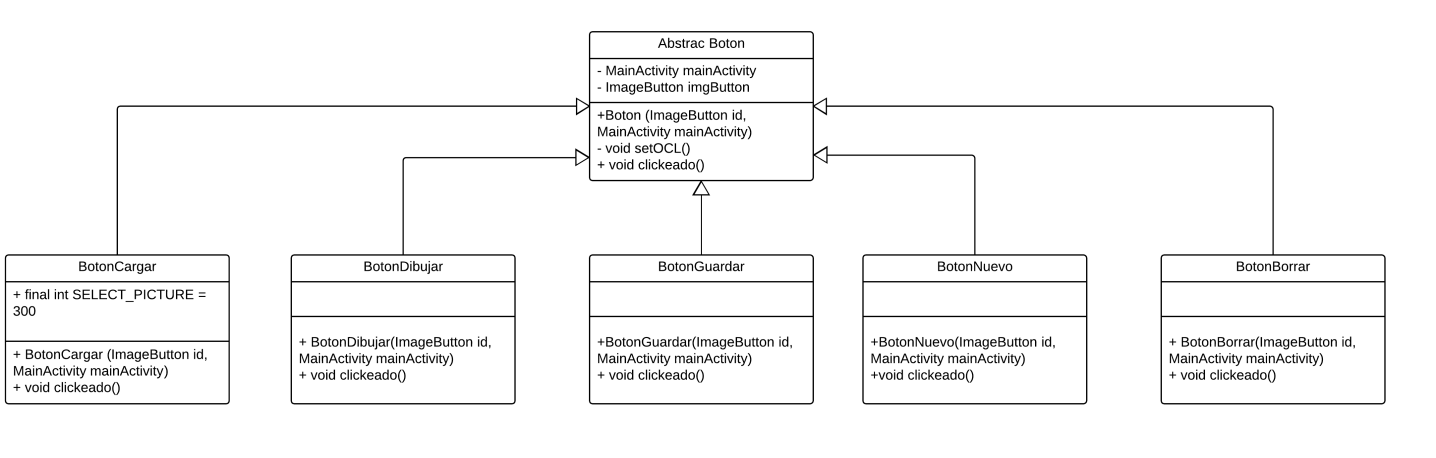
**Patrón Observer**

Este patrón de diseño define una dependencia del tipo *uno-a-muchos* entre objetos, de manera que cuando uno de los objetos cambia su estado (Subject), notifica este cambio a todos los dependientes (Observador). Se trata de un *patrón de comportamiento* (existen de 3 tipos: Creación, Estructurales y de Comportamiento), es decir, está relacionado con algoritmos de funcionamiento y asignación de *responsabilidades* a clases y objetos.

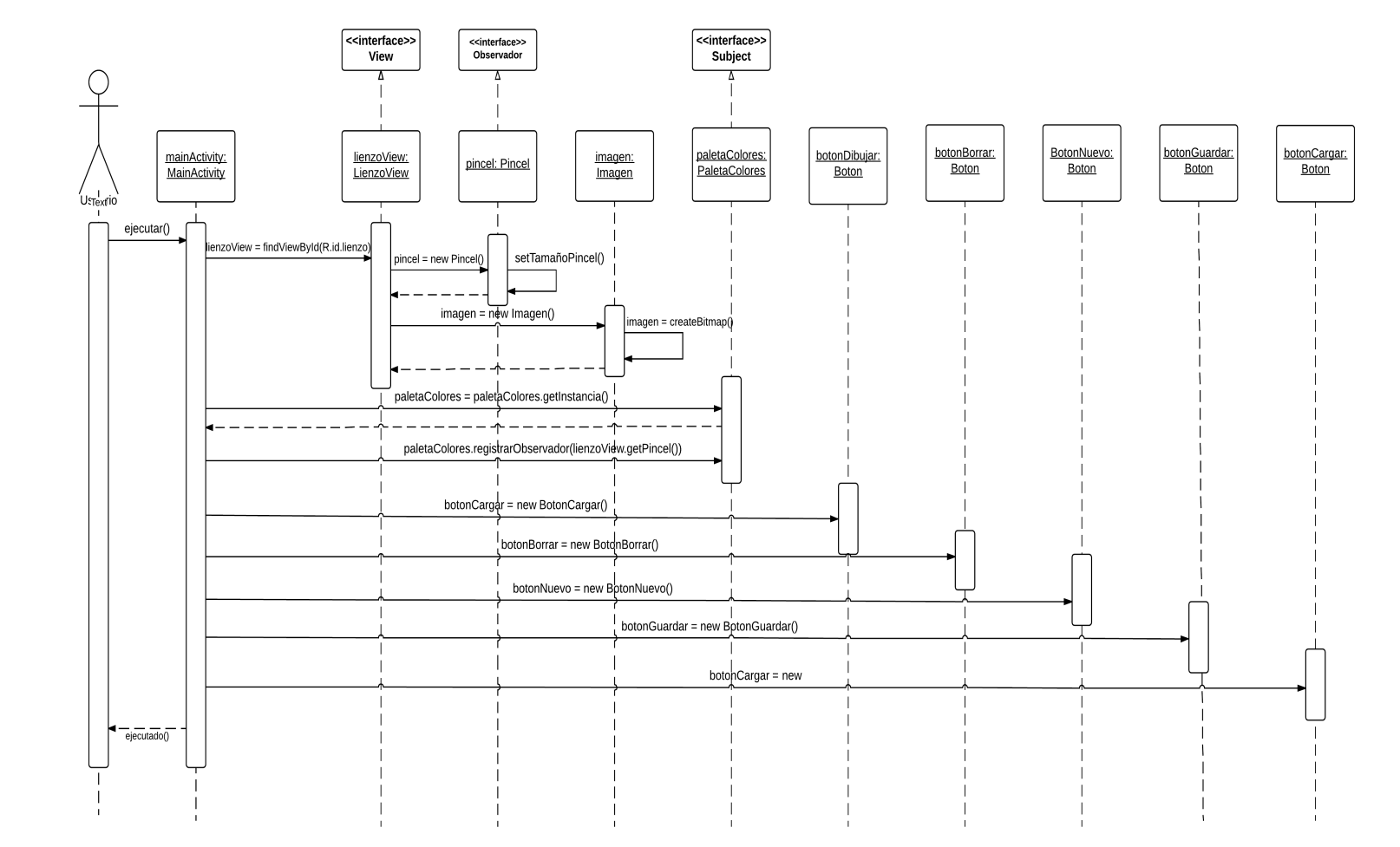
Por esto mencionado anteriormente, nosotros sabemos que en nuestra aplicación tendremos la Paleta de Colores que será el sujeto observado y el pincel, el observador, ya que necesitamos que cuando hay un evento en la paleta o actualización, el pincel actualice el color al seleccionado por el usuario en la misma.

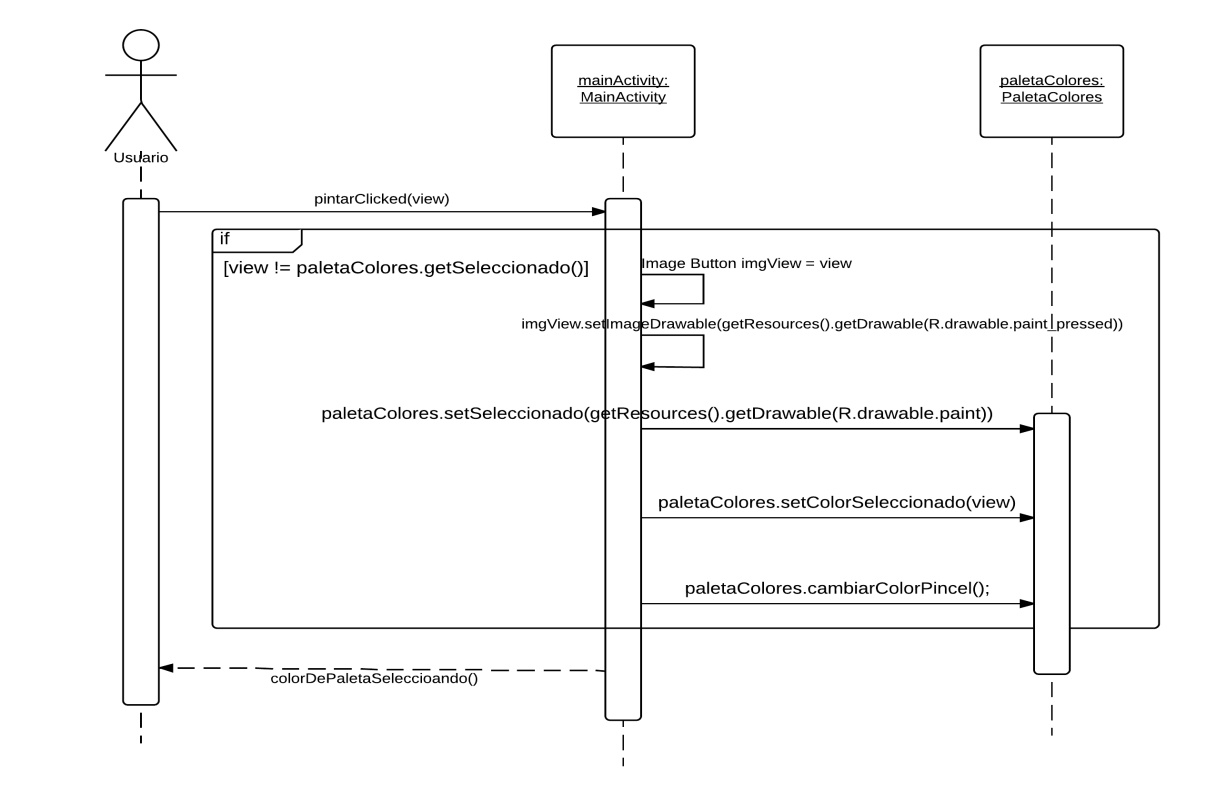
  
**Patrón Strategy**

Este patrón se encarga de controlar el intercambio de mensajes entre diferentes objetos para resolver un problema.Todos las clases parecidas, se encapsulan como subclases de una superclase, es decir, heredan los atributos y métodos de su clase padre. En el caso de nuestra aplicación, como todos los botones deben reaccionar frente a un evento, implementaremos el método clickeado en la clase abstracta Botón, y haremos que todos los botones hereden de esta clase. Así nos aseguramos que todos los botones tengan este método implementado.

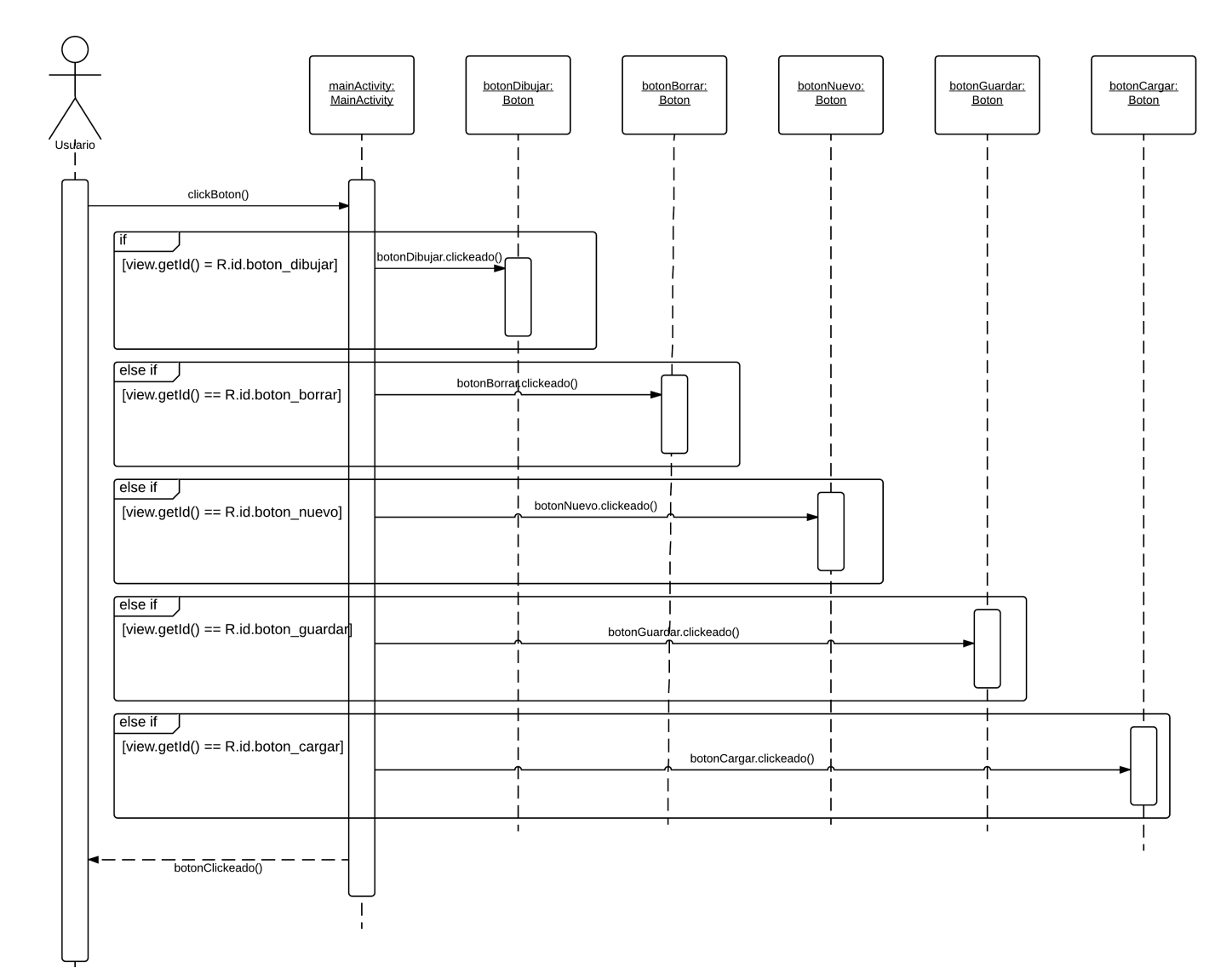
****

**Diagramas de secuencia**

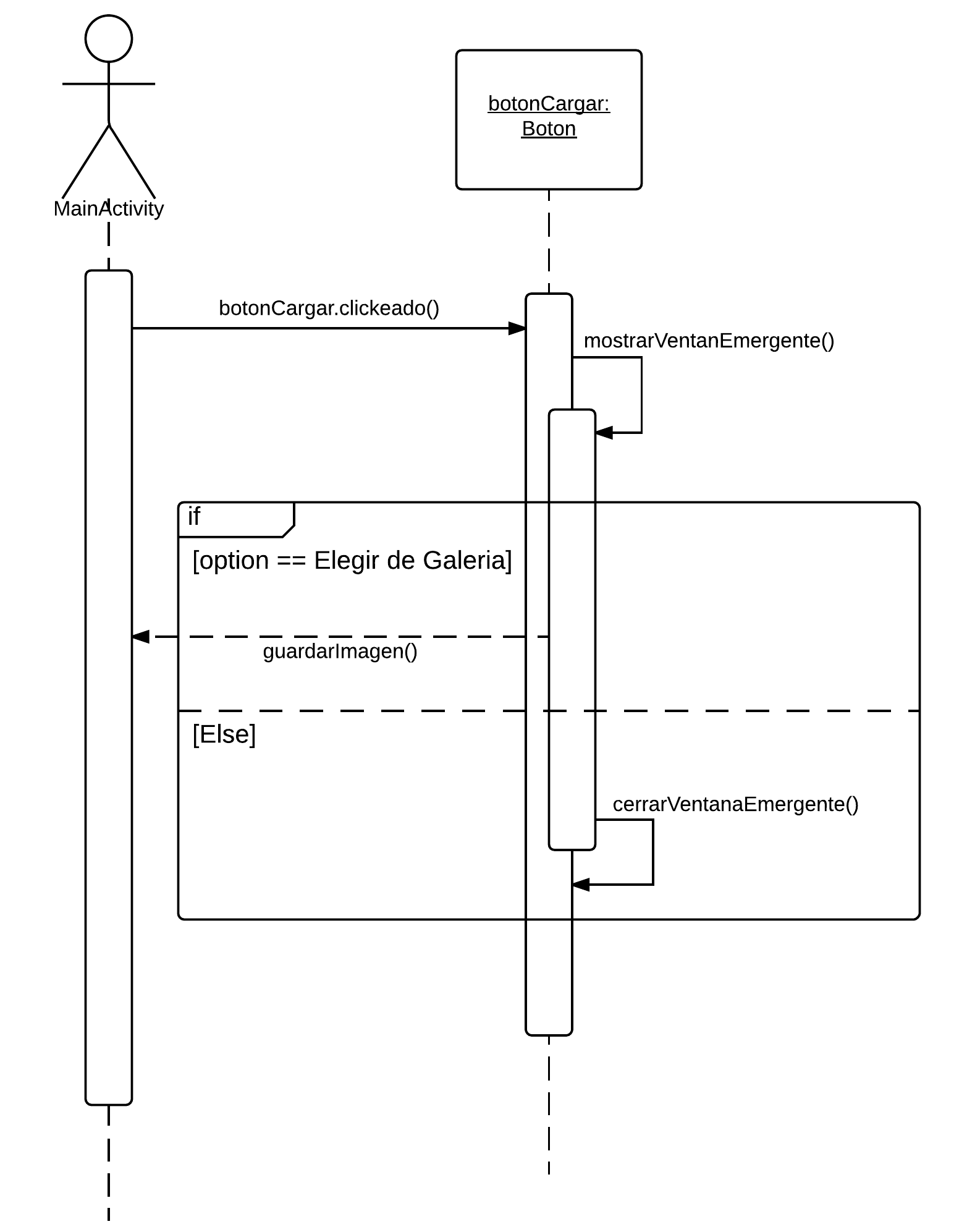
**Secuencia de inicio**

**Secuencia cuando el usuario selecciona la paleta de colores**

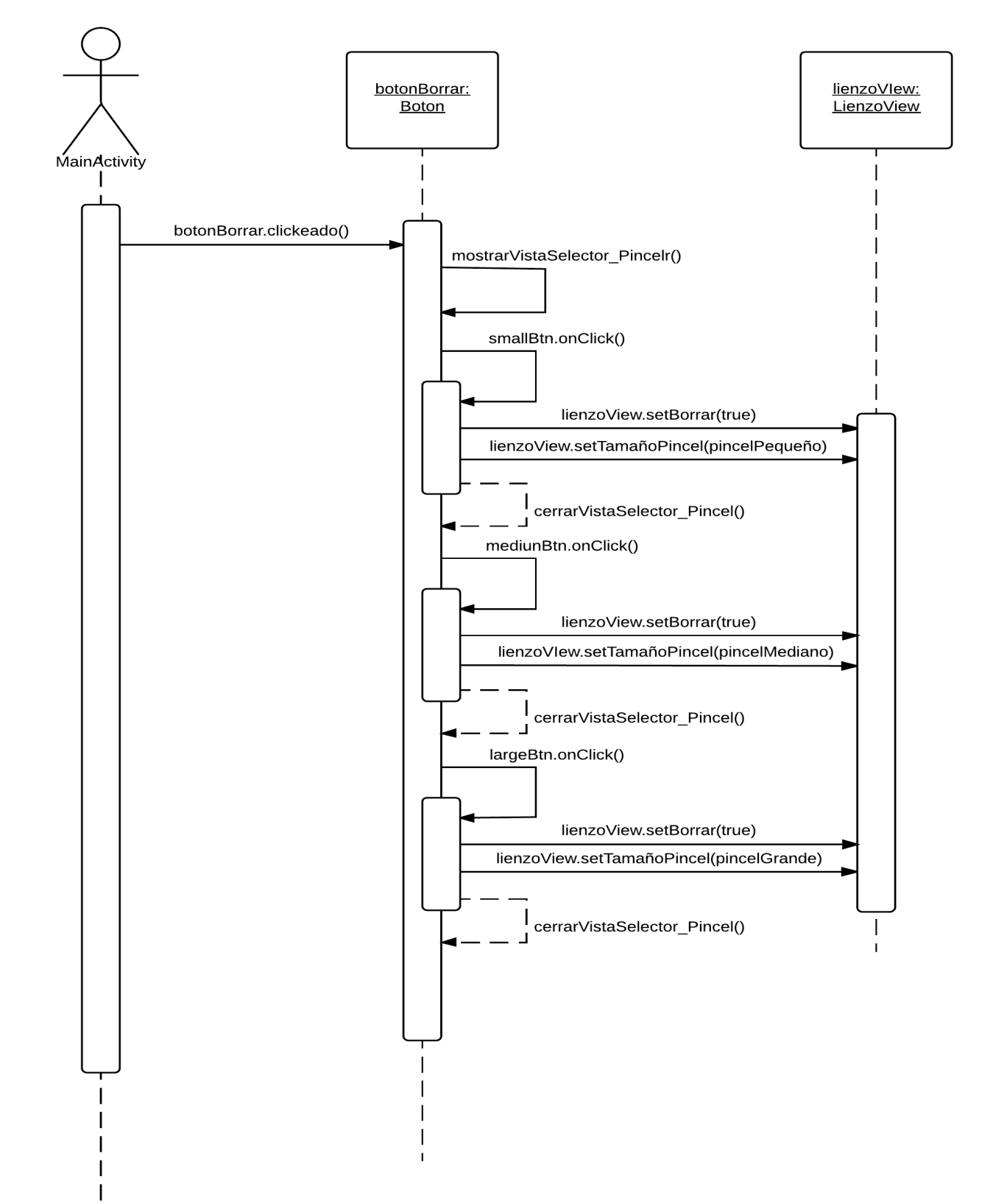
**Secuencia cuando el usuario selecciona un botón**

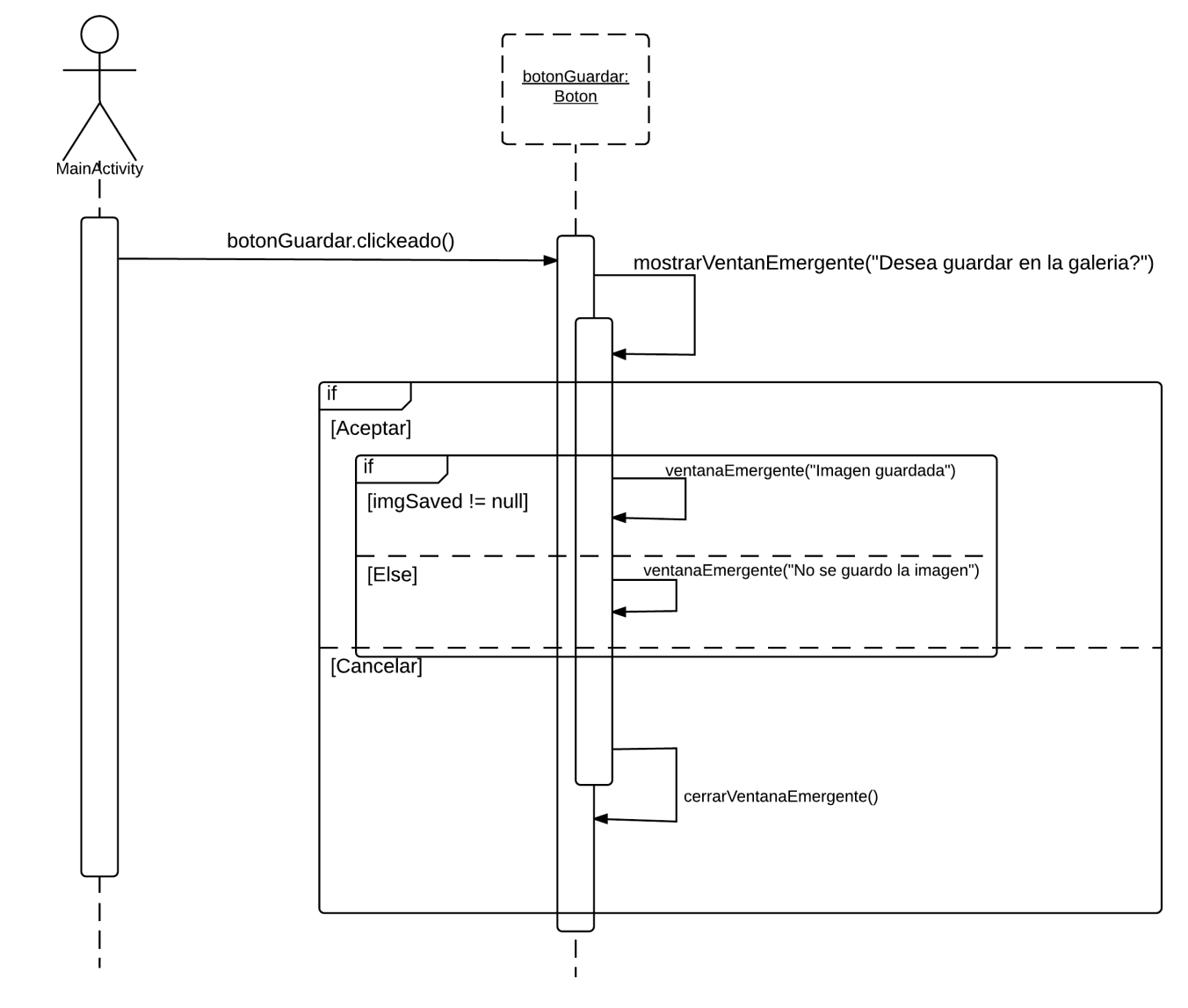
****

**Secuencia cuando el usuario selecciona el botón cargar**

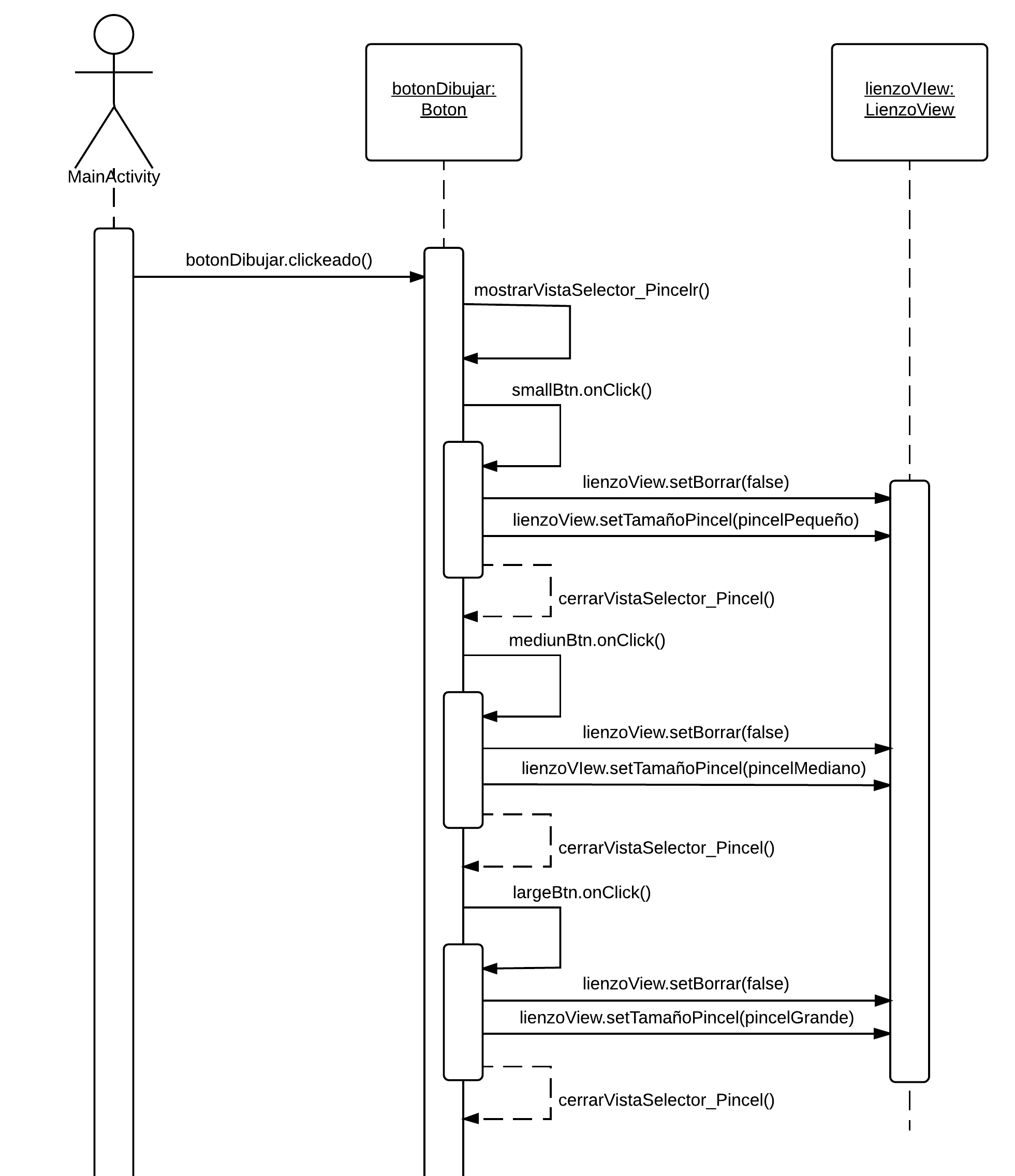


**Secuencia cuando el usuario selecciona el botón borrar**

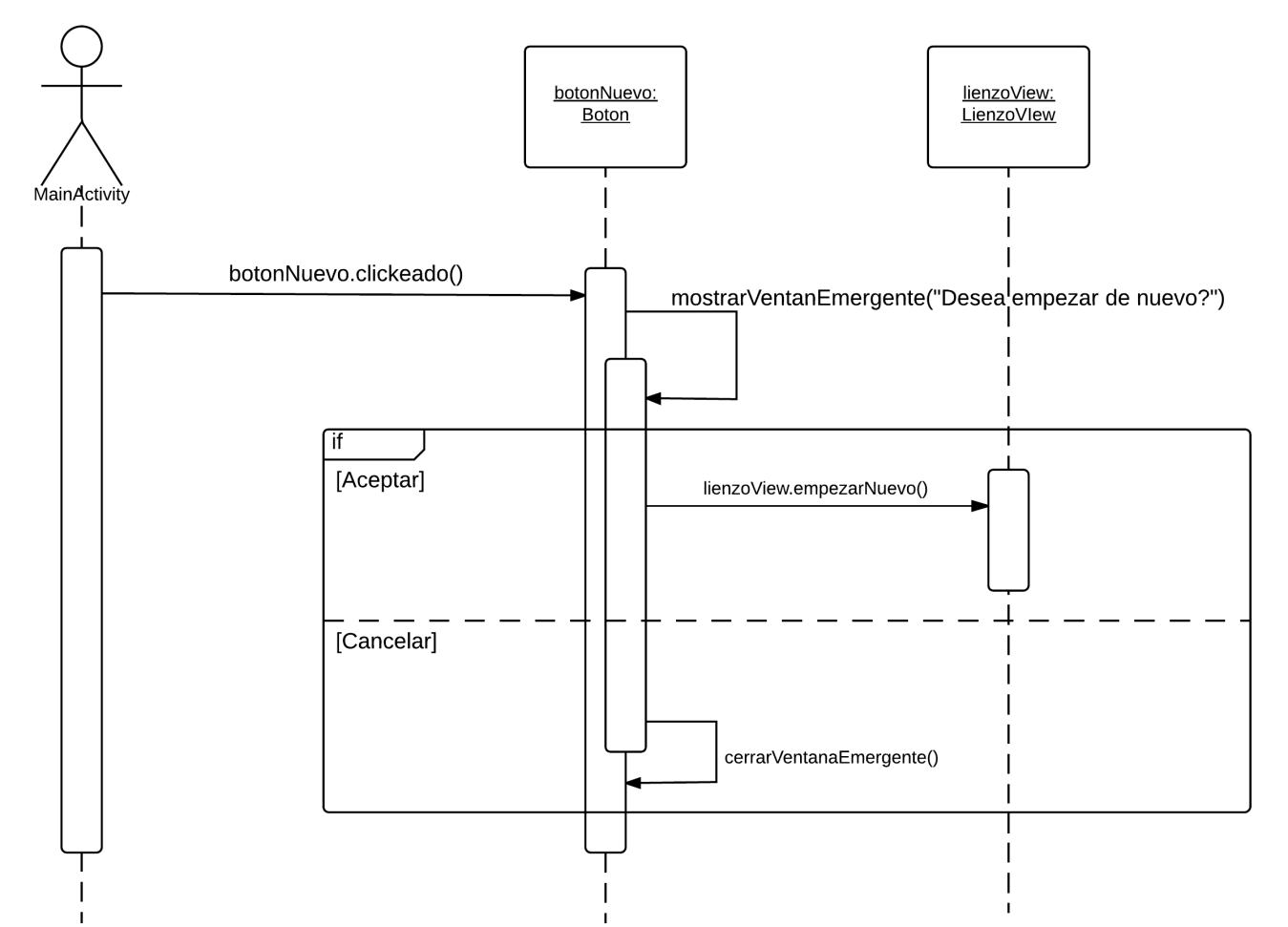


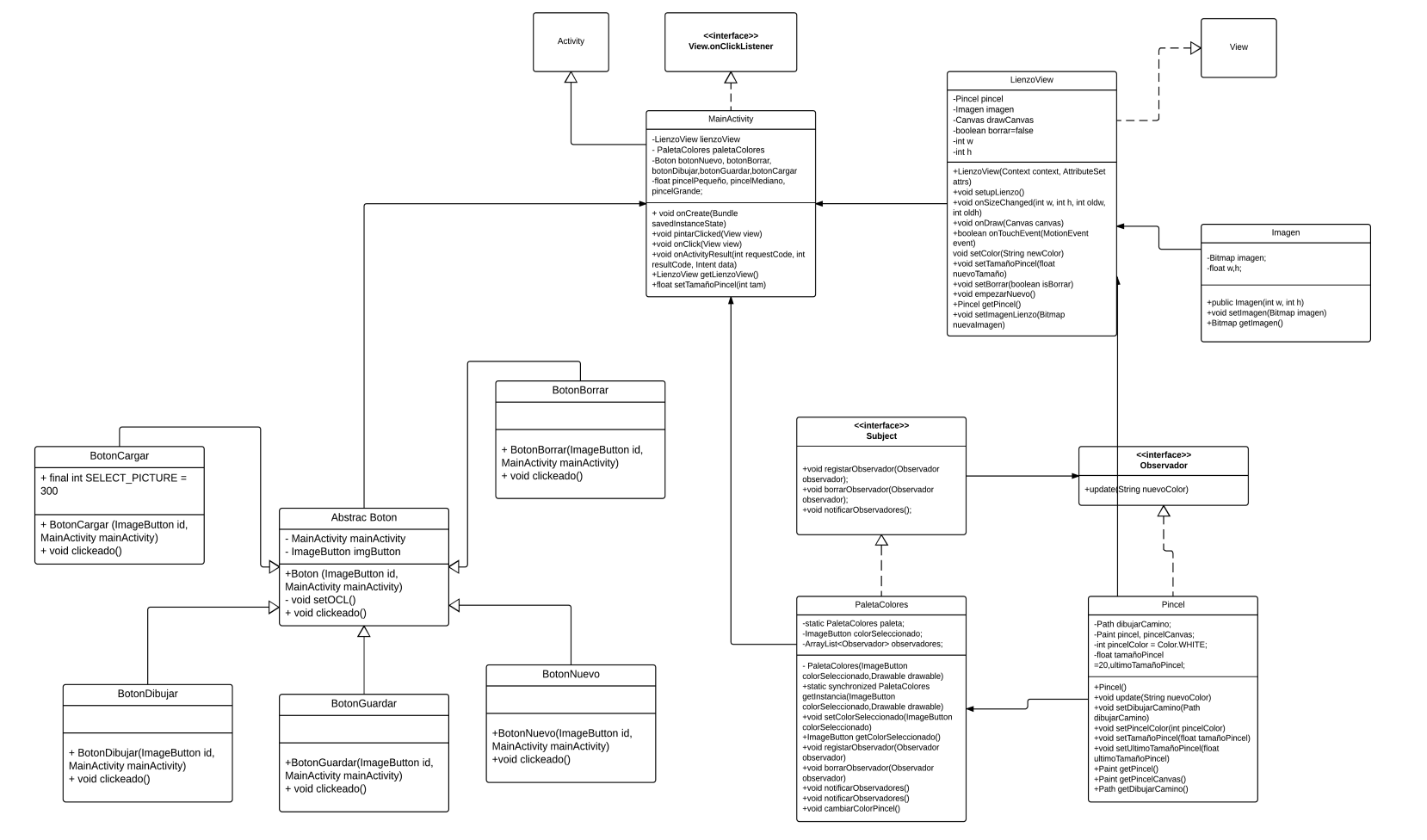
**Secuencia cuando el usuario selecciona el botón guardar**

**Secuencia cuando el usuario selecciona el botón dibujar**

****

**Secuencia cuando el usuario selecciona el botón nuevo**

****

**Diagrama de clases**

(Hacer zoom, en caso de no observarse correctamente)

**Historial de versiones de documentos**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre | Versión | Fecha | Observaciones | Autor |
| Administración-de-las-configuraciones | 1.0.0 | 08/05/2017 | Documento de administración de las configuraciones completo(Entregado) | Gon, Alexander |
| Requerimientos-funcionales-del-sistema | 1.0.0 | 08/05/2017 | Documento de requerimientos del sistema(Entregado) | Orecchini, Stefano |
| Requerimientos-funcionales-del-sistema | 1.1.0 | 21/06/2017 | Actualizo:  -Requerimientos-Pruebas de sistema  -Diagrama de Casos de Uso  -Matrices de trazabilidad | Lenta, Luis |
| Diseño de la arquitectura | 1.0.0 | 22/06/2017 | Documento con el diseño de la arquitectura | Lenta, Luis |
| Diseño e implementación | 1.0.0 | 23/06/2017 | Documento con el diseño y la implementación | Gon, Alexander |
| Requerimientos-funcionales-del-sistema | 1.1.1 | 25/06/2017 | Agrego diseño preliminar de la arquitectura | Lenta, Luis |
| Diseño de la arquitectura | 1.1.0 | 26/06/2017 | Merge con Diseño e implementación | Orecchini, Stefano |