# Requirements

# Git workflow

1. Set up a Repository: To get started, create a shared Git repository that will host the project's code. It's important to initialize it with a README, .gitignore, and a License.
2. Develop in Branches: Each developer should create a new branch for each new feature or bug fix. This keeps changes organized and separated until they're ready to be combined with the main code.
3. Naming Conventions: Use clear and precise naming conventions for your branches, e.g., 'feature/cards-dealing', 'bug/queen-detection'.

## Branches

1. **feature/card-dealing**: This branch could be used for the implementation of distributing the game cards to the players.
2. **feature/card-identification**: This branch could house the code necessary to identify which cards the players have received.
3. **feature/pair-detection**: This branch can be used for creating the algorithm that finds pairs in the player's cards.
4. **feature/three-of-a-kind-detection**: This branch would be used for detecting the three of a kind or "terna".
5. **bugfix/card-duplication**: If there's a bug where the same card is dealt more than once, such a bug could be fixed in this branch.
6. **feature/game-rules-logic**: This branch could be utilized to implement the rules logic of the game.
7. **feature/player-interface**: To create the interface that the player interacts with.
8. **feature/score-calculation**: To implement the logic of how the score will be calculated based on the cards.
9. **feature/game-result-display**: To display the end result of the game.
10. **bugfix/card-detection-issues**: If there are bugs in card detection, they could be fixed here.

# Swing Threading

Swing Threading se refiere al modelo de concurrencia utilizado por la biblioteca de Swing en Java, en donde toda la **interacción con los componentes de Swing** debe ocurrir en un hilo especial llamado **Event Dispatch Thread (EDT).**

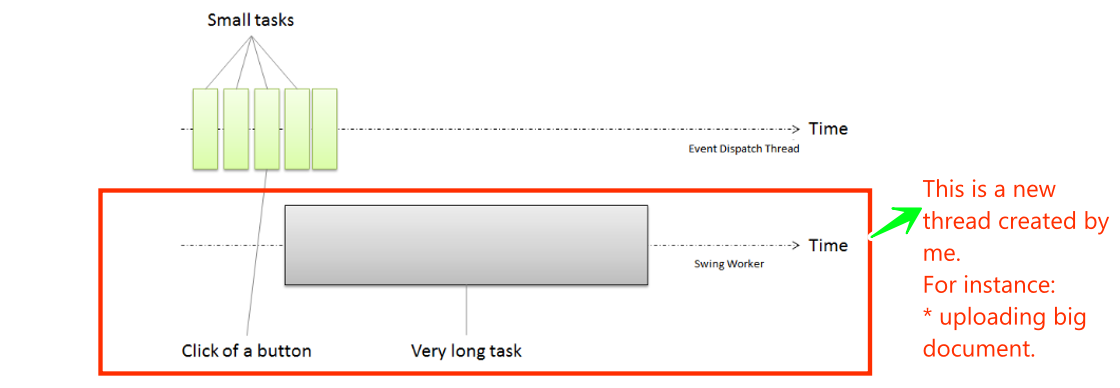
## Event Dispatch Thread

If there be short and long events in Event Dispatch thread like below, the application can't be responsive:

A long rectangular object with text

Description automatically generated

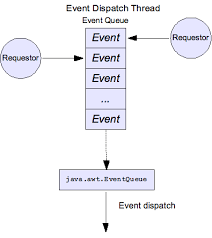
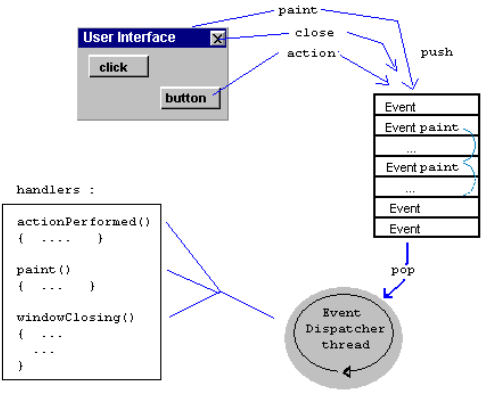
For the Sake of responsiveness in Swing, Event Dispatch thread should only be used for short events. while long events should be executed on SwingWorkers.

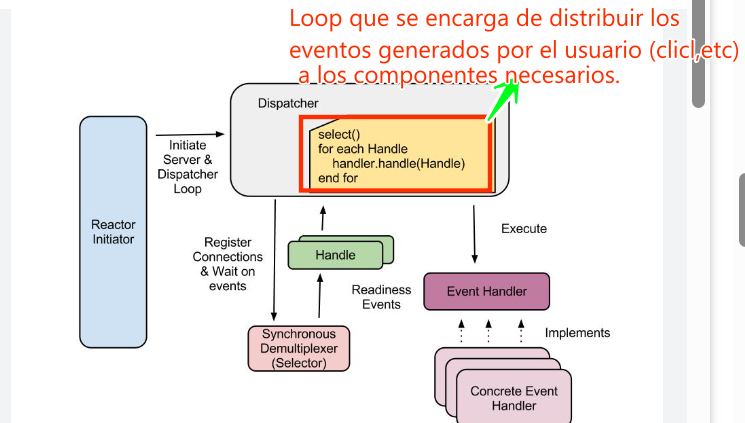


Imagine that there is a lot of short events.

A diagram of a task

Description automatically generated





The complete Swing processing is done in a thread called **EDT (Event Dispatching Thread)**. Therefore you would block the GUI if you would compute some long lasting calculations within this thread.

The way to go here is to process your calculation within a different thread, so your GUI stays responsive. At the end you want to update your GUI, which **have to** be done within the EDT.

Now EventQueue.invokeLater comes into play. It posts an event (your Runnable) at the end of Swings event list and is processed after all previous GUI events are processed.

Be careful **not** to update your Swing GUI from a different thread. In most cases this produces some strange updating/refreshing issues.

Swing Threading se refiere al modelo de concurrencia utilizado por la biblioteca de Swing en Java, en donde toda la interacción con los componentes de Swing debe ocurrir en un hilo especial llamado Event Dispatch Thread (EDT).

El Event Dispatch Thread (EDT), es un loop que se encarga de manejar los eventos generados por la interacción del usuario (tales como clics de mouse o pulsaciones de tecla) y repartir estos eventos a los componentes correspondientes para ser procesados. Las actualizaciones de la interfaz gráfica de usuario y las tareas que corren en este hilo no debería bloquearse por tareas largas, ya que ello haría que la interfaz gráfica de usuario parezca congelada.

El método **EventQueue.invokeLater(Runnable)** se utiliza para colocar una tarea en la cola del Event Dispatch Thread. Cuando este hilo tiene tiempo, corre la tarea. Esto asegura que la tarea se ejecutará de manera segura sin interrumpir el Event Dispatch Thread mientras está procesando otros eventos.

Un escenario típico donde se usa **invokeLater()** es para realizar una tarea que modifica la interfaz gráfica de usuario después de haber realizado alguna operación de larga duración en un hilo separado. Este método nos permite "retrasar" esta tarea hasta que el Event Dispatch Thread esté listo para manejarla.

Aquí hay un ejemplo simple de cómo podría verse en código:



En resumen, Swing Threading se trata de cómo se maneja la concurrencia en Swing, utilizando un hilo especial para tratar todos los eventos y actualizaciones de la interfaz gráfica de usuario y el método **invokeLater** se usa para asegurar que las tareas se ejecutan en este hilo de manera segura.

## Analogía

Imagina que estás en una cocina. El cocinero principal (Event Dispatch Thread) está muy ocupado preparando platos (realizando tareas). Cada vez que el cocinero principal acaba de preparar un plato, coge la próxima receta de la pila de recetas (EventQueue) y comienza a trabajar en ella.

Supón que tú eres un ayudante en esta cocina, y acabas de preparar un nuevo ingrediente (una tarea que has realizado en un hilo separado). No puedes simplemente interrumpir al cocinero principal y entregarle el ingrediente, porque podría estar a medio camino de añadir especias a un plato delicado.

Entonces, en vez de hacer eso, dejas tranquilo al cocinero y pones el nuevo ingrediente en su pila de recetas a espera (EventQueue). Esto es lo que hace **invokeLater()**. Le permite al cocinero principal (EDT) terminar lo que está haciendo y luego, cuando tiene tiempo, cogerá la próxima receta (tu tarea) de la pila (EventQueue) y comenzará a trabajar en ella.

En este caso, las "recetas" son tareas que necesitan ser ejecutadas y los "ingredientes" son actualizaciones a la interfaz gráfica de usuario.

Entonces, en términos simples, **invokeLater()** es una forma segura de decir "Hola, tengo una tarea que necesita ser ejecutada, pero sé que estás ocupado, así que cuando tengas tiempo, ¿puedes encargarte de ella, por favor?" a EDT.

## Funcionamiento

Cuando se inicia la aplicación, el EDT (Event Dispatch Thread) comienza a correr en un loop infinito, esperando por eventos de la cola de eventos (EventQueue), que inicialmente está vacía.

1. El usuario hace click en el botón "Abrir archivo". Este evento se pone en la cola de EventQueue.
2. Cuando el EDT tiene tiempo, toma este evento de la cola y comienza a ejecutar la operación de abrir el archivo.
3. Mientras el archivo se está cargando, imaginemos que el usuario hace click repetidamente en el botón de "Hola". Cada uno de estos clicks se convierte en un evento que se pone en la cola de EventQueue.
4. Pero el EDT todavía está ocupado cargando el archivo. Para no bloquear la aplicación, la operación de apertura del archivo se debería hacer en un hilo separado. Con ello, el EDT puede volver a su trabajo principal de manejar los eventos de la interfaz de usuario.
5. Ahora, el EDT tiene tiempo para manejar los eventos de clicks en el botón de "Hola". Toma cada click de la cola en el orden en el que llegaron, y muestra el mensaje "Hola" para cada uno.
6. Cuando el hilo separado ha terminado de cargar el archivo, usa EventQueue.invokeLater() para poner una tarea en la cola que actualiza la interfaz de usuario, digamos mostrando el contenido del archivo.
7. Finalmente, cuando el EDT vuelve a tener tiempo, toma esta tarea de la cola y actualiza la interfaz de usuario.

## Explicación final

Muy buena pregunta. La principal razón es que la biblioteca Swing no es "thread-safe". Esto significa que no puedes simplemente manipular sus componentes desde cualquier hilo.

Todos los componentes de Swing deben ser accedidos desde un hilo específico llamado Event Dispatch Thread (EDT). Cuando haces clic en un botón, mueves una ventana o realizas cualquier otra acción en la interfaz gráfica, esa acción se maneja en el EDT.

El siguiente código:

A screen shot of a computer code

Description automatically generated

Cuando ejecutas main(), en realidad se está ejecutando en el hilo "principal" de Java, no en el EDT. Por lo tanto, estás accediendo a los componentes de la interfaz gráfica desde un hilo que no es el EDT, lo que puede causar problemas.

Por otro lado, este código:

A computer code with colorful text

Description automatically generated

Hace que el código se ejecute en el EDT mediante el uso de `EventQueue.invokeLater()`. Asegura que JFrame y todos sus componentes se inicializan en el EDT, lo cual es más seguro y elimina el riesgo de problemas de concurrencia.

Entonces, en resumen, siempre debes interactuar con los componentes de Swing en el EDT. El método `EventQueue.invokeLater()` nos proporciona una forma fácil y segura de hacer esto.

# Escalera

* *Que muestre el mensaje en la pantalla con las imágenes.*
  + Si yo tengo un array con toda la información de las cartas.
  + Agrupar por pinta las cartas. Map<Pinta,List<Integer>>
    1. Verificar que la lista sea mayor o igual q 3
    2. Ordenar la lista
    3. Verificar Secuencia:
       1. Si el 1er es (igual al 2do valor ó 2do valor+1).
       2. Si el 2do es (igual al 3er valor ó 3er valor+1)
       3. Si el 3ro es (igual al 4to valor ó 4to valor+1)