3 DE MARZO DE 2023

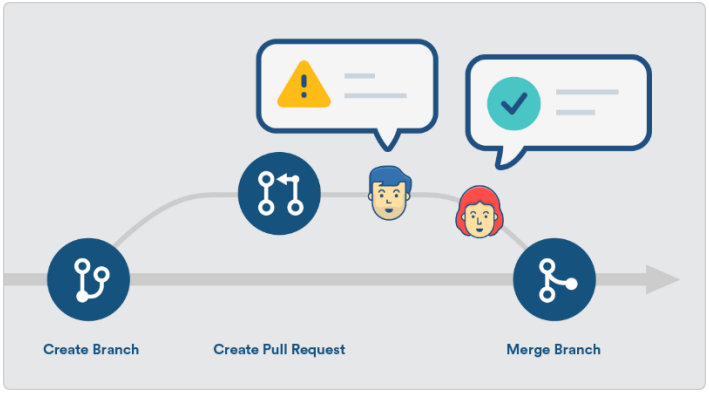
SEMANA 2 EVALUACION

JESÚS ENRIQUE PALOMARES MENDOZA

JESUSPALOMARES03@GMAIL.COM
ACCENTURE
XIDERAL

Git (Pull Request)

Es una herramienta que permite a los integrantes de un equipo de desarrollo colaborar en un proyecto. Esto se realiza con la acción de enviar un a propuesta de cambio, siendo esta revisada por el equipo.



Este proceso comienza cuando un integrante del equipo crea una rama a partir de la rama principal, una ves que el proyecto haya sido modificado este se guarda en la rama secundaria y se procede a notificar de los cambios.

Si los cambios son aceptados, serán fusionados a la rama principal del proyecto a efecto de enriquecerlo.

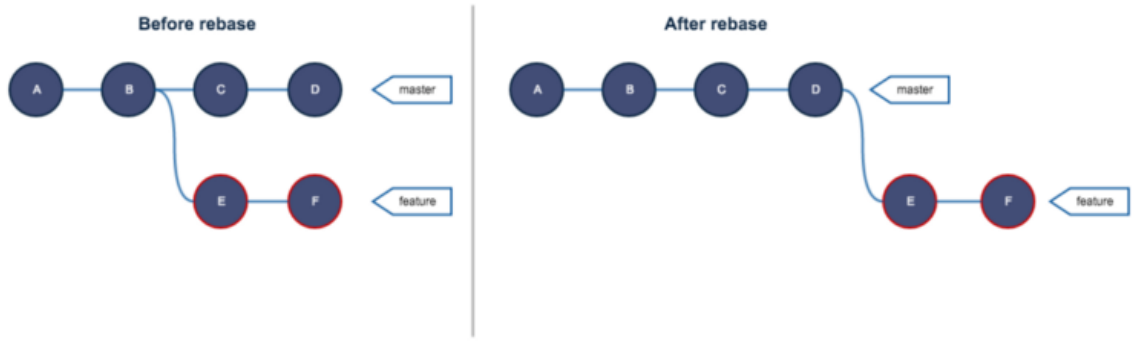
En resumen, un Pull Request es un Mecanismo de colaboración donde los integrantes del equipo realizan sus modificaciones y se ponen a validación para ser agregados en el proyecto.

Git (Fork)

Un fork es crear copia de un reposito para poder realizar libremente nuestros cambios; estos cambios no afectan a repositorio original, ya que al realizar un fork se crea un nuevo repositorio en nuestra cuenta.

Esta herramienta permite que un número de integrantes de un equipo trabaje sin afectar a los demás. Una vez terminado se puede compartir los cambios mediante un pull request.

Git (Rebase)

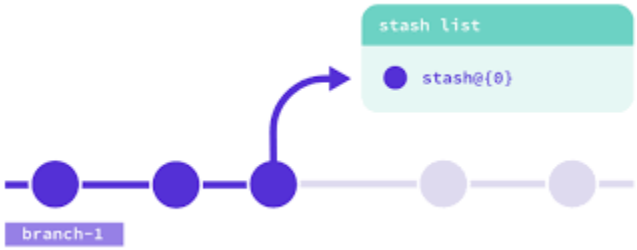


A diferencia de Merge, que combinan los commits de la rama secundaria, y posteriormente los agrega a la rama principal, **Rebase** lo que hace es igualar la rama secundaria con los commits ya creados previamente a la creación de la rama secundaria, agregar los nuevos commits y continuar usando la rama. A esto también se le conoce como MERGE FAST FORWARD.

En resumen, es mover los commits de una rama a otra; tomando los commits de la rama secundaria y los aplica a la rama principal como si fueran creados directamente.

Git (Stash)

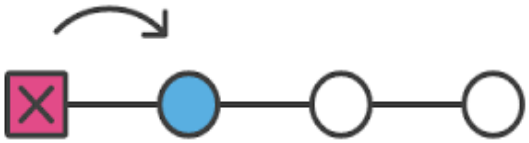
Este comando permite guardar cambios de nuestro proyecto de manera temporal, es decir sin crear un nuevo commit, con la funcionalidad de cambiar de tarea.



Un ejemplo, es cuando nos encontramos trabajando en un cambio principal ya sea implementar un nuevo método o simplemente configurar unos parámetros, y en esos momentos nos piden realicemos otra actividad fuera de la que nos encontramos realizando, para ello realizamos un stash que nos permite guardar de manera temporal nuestro avance para así cambiar a otra actividad diversa, sin necesidad de esperar a terminar el proceso anterior.

Git (Clean)

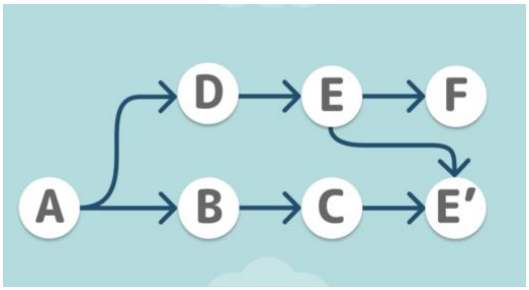
Este comando tiene como función borrar carpetas que no son parte de nuestro directorio de trabajo, carpetas que se mantienen después de realizar un git rest --hard (Revertir cambios), incluso carpetas o archivos que son agregados sin un seguimiento en el directorio. Básicamente se encarga de eliminar archivos que no son necesarios para el proyecto.



Git (Cherry Pick)

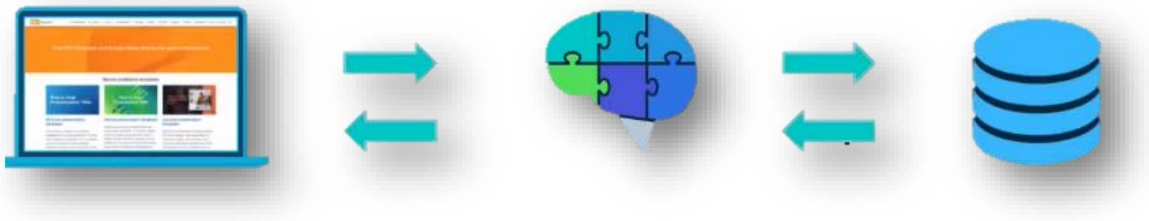
Es una herramienta que permite aplicar un commit especifico de ramas secundarias a la rama principal, para ello se tiene que especificar que commit se desea copiar y a que rama se aplicara.

Este comando es funcional cuando queremos aplicar cambios específicos sin tener que fusionar toda la rama generada, evitando crear un merge o rebase.



Modelo Vista Controlador (MVC)

Este modelo es un patrón de diseño de software el cual tiene como funcionalidad la solución general a problemas comunes, este método viene siendo una especie de plantilla para organizar la lógica del proyecto, siendo este el patrón de diseño mas utilizado dentro del ecosistema de desarrollo web.



Dicho modelo separara la aplicación en 3 componentes lógicos siendo; modelo el encargado de almacenar e administrar los datos con lo que se trabajara, estos datos son obtenidos por medio de nuestra base de datos.

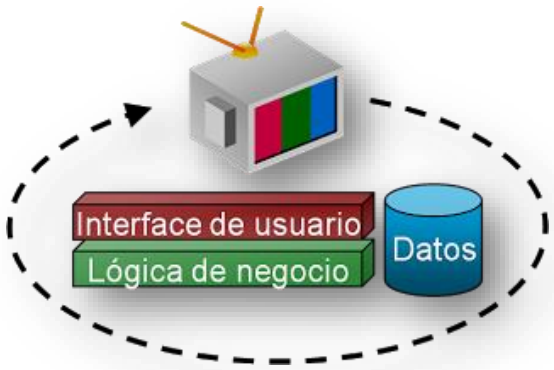
La vista o view es la encargada de mostrar una representación visual de la información al usuario por medio de una interfaz gráfica. Esta parte también es la encargada de mostrar los datos al usuario de una manera comprensible desde cualquier dispositivo.

Por último, tenemos el controlador o controller; el cual es el encargado de hacer que la vista y el modelo trabajen entre si por medio de instrucciones.

La separación de este modelo permite que cada una de las partes pueda ser modificada sin afectar a las demás, sin olvidar que esta separación de partes debe estar bien definida entre la capa de presentación y lógica de negocio.

Aplicacion Monolitica vs Microservicios

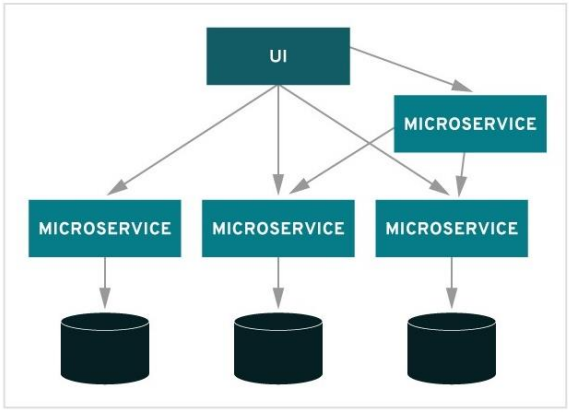
La arquitectura monolítica ha sido por muchos años la estructura predilecta para el desarrollo de software la cual consiste en una interfaz de usuario del lado del cliente, una aplicación del lado del servidor y una base de datos.



Esta arquitectura tiene ventajas principales como son; fácil de testear y de buggear, fácil de desplegar y fácil de desarrollar.

Sin embargo, así como tiene ventajas también tiene desventajas y entre ellas se encuentra que en dicha arquitectura es difícil incorporar nuevas tecnologías, su escalabilidad es costosa y al ser trabajado todo dentro de una clase queda un código muy grande veces difícil de comprender, siendo así para realizar un cambio se tiene que adentrar a todo el proyecto.

En 2011 surge el termino microservicios como alternativa a monolíticos, los microservicios dividen la aplicación en unidades independientes, cada una de estas unidades cuenta con su lógica independiente y su base de datos, estas se comunican entre si a través de un API.



Dentro de sus ventajas cuenta con una escalabilidad efectiva ya que cada microservicio es escalado de manera independiente y también se pueden ocupar múltiples tecnologías es decir cada micro servicio puede estar desarrollado en diferentes lenguajes.

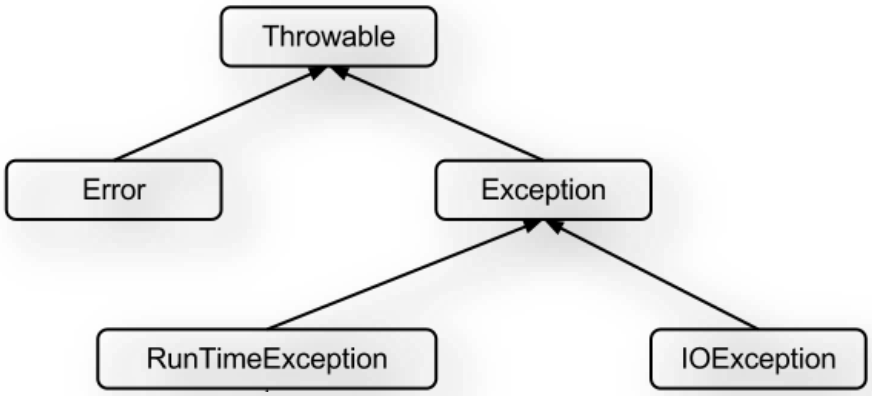
Esta arquitectura conlleva mayor complejidad ya que se debe diseña como se conectarán los componentes entre sí.

¿Cuándo usar cada una?; cuando un proyecto es sencillo y se necesita de manera rápida lo más conveniente es que sea desarrollada desde la arquitectura monolítica, y cuan el proyecto es grande y se requiere que sea escalada con el tiempo lo mejor es utilizar microservicios.

Tipos de excepciones

Una excepción es cuando un programa viola las restricciones semánticas, cuando esto sucede comunica un error, esto se comunica a través de la máquina virtual, muchos tipos de errores pueden provocar una excepción, desde un desbordamiento de memoria, el intento de dividir por cero o intentar acceder a un vector fuera de sus límites. Cuando ocurre una excepción nuestro programa se detiene sin continuar con el resto del código.

Jerarquía de Excepciones



La clase throwable deriva directamente de la clase objet, la cual es la super clase que de todos los errores o excepciones que pueden salir de los programas, esta clase contiene dos clases hijas; error y exception.

La clase **Error** tiene que ver con las fallas de hardware, son las alertas o problemas que se generar de forma externa al software, y los cuales no cuentan con una solución directamente en código dichos problemas no tienen que ver con el programador.

La clase **exception** se divide de dos tipos los IOException o mejor conocidas como excepciones verificadas, también contamos con la clase RunTimeException o conocidas como Excepciones no verificadas.

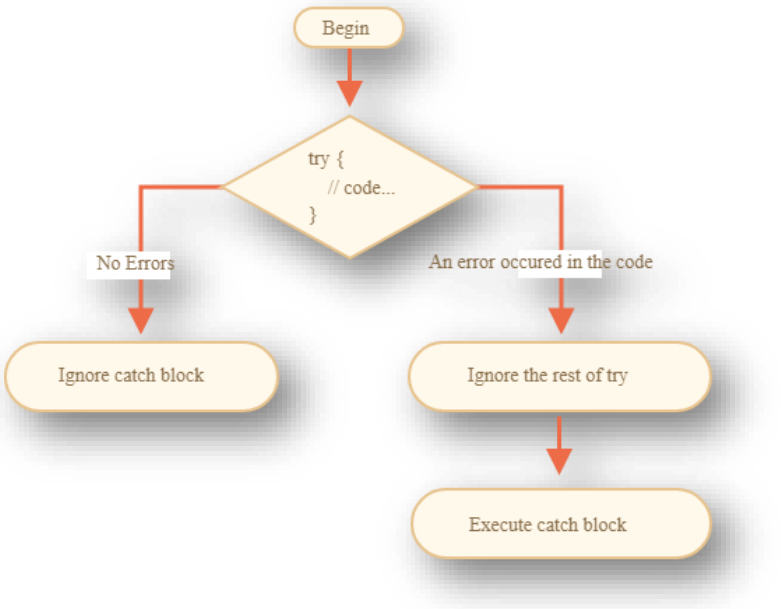
Las IOException no dependen directamente del programador, pero si se puede solucionar, dichos errores se basan en las entradas y salidas del programa, un ejemplo sencillo vendría siendo cuando

una ruta donde se encuentra un archivo de texto ha cambiado, al suceder esto genera una excepción verificada que no depende del programador.

Run Time Exception son errores que, si dependen del programador, son pequeñas fallas que se generan cuando se invalidad reglas ya sean lógicas o de sintaxis, estos errores deben ser corregidos de lo contrario nuestro programa de detendrá cuando una de estas sea detectada.

Multi Catch

Try Cachar se utiliza cuando encontramos excepciones verificadas, aquellas generadas por el IOException, dentro del try es la parte del bloque donde puede ocurrir la excepción dejado posteriormente el catch donde en caso de ejecutarse una excepción esta realizara lo que se le indique.



Lo que realiza el multi catch es que cuando llega al menos una excepción de un múltiple de opciones, entonces ejecuta lo que se le indique, en otras palabras, si existe al menos una excepción entonces ejecuta el catch.

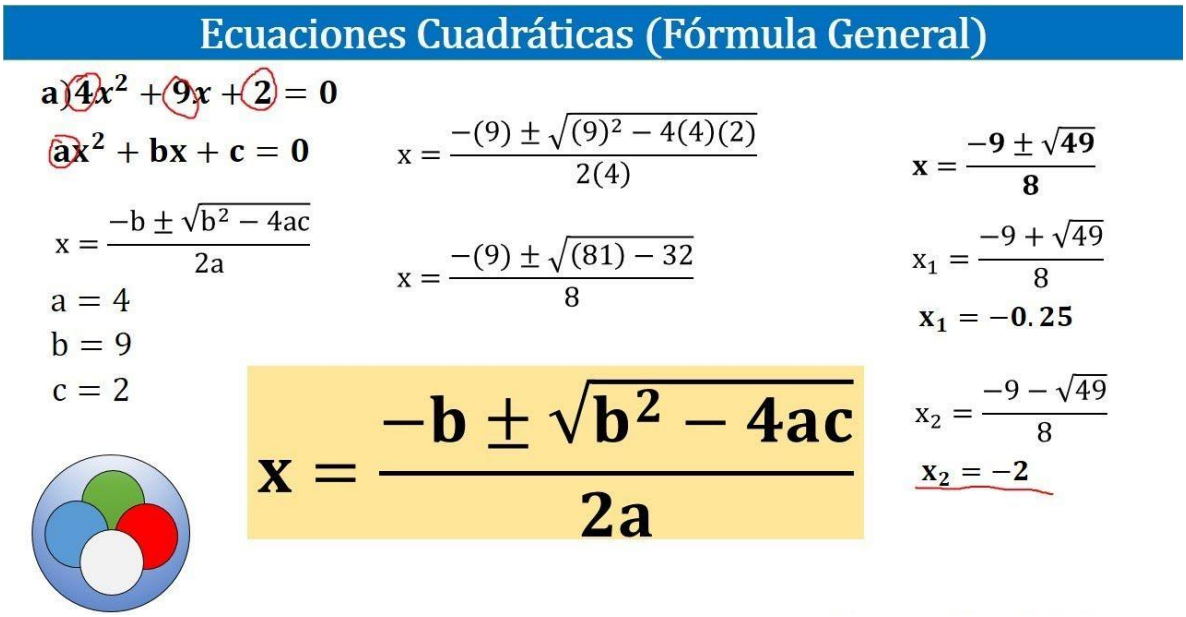
TRY WITH RESOURCE

En try catch finally se tenía que ejecutar un bloque de código infaliblemente tras la finalización del bloque catch, en try with resource lo que se realiza es que un recurso que ocupemos sea automáticamente cerrado después de ser utilizado, todo esto es independientemente de si se haya o no generado una excepción.

Unas de las ventajas que asegurar que los recursos sean cerrados evitando problemas como fugas de memoria o bloqueos.

Junit

Dentro de nuestras pruebas en Junit, se realizo un test donde ponemos en ejecuacion las operaciones básicas con el problema de una ecuación cuadrática bajándonos en el siguiente ejemplo:



Donde los valores utilizados dentro de nuestros tests son a=4, b=9 y c=2.