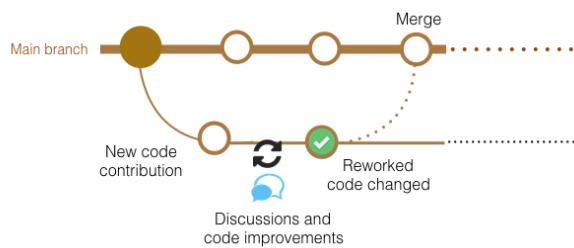


# Semana Dos

Fecha	27 febrero - 3 marzo 2023
Nombre	Olivia Yuyu Maceda Pérez

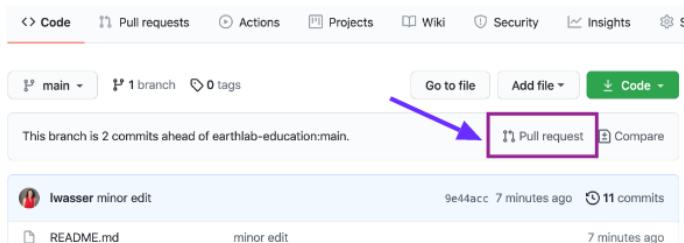
## Ejercicios Xideral

- **Explicar**
  - **pull request**



Simplified Pull Request process

Es cuando un colaborador esta listo para iniciar el proceso de mezclar su nuevos cambios del código con la rama principal donde se encuentra el repositorio. Es decir, manda un pull request solicitando que el 'encargado' extraiga una rama de su repositorio en el repositorio principal.



Solicitud para que tu código se fusione con el código del repositorio donde has colaborado.

#### 4 PUNTOS IMPORTANTES

- Repositorio de origen
- Rama de origen
- Repositorio destino
- Rama destino

Una vez que el pull request se abre, el encargado, colaboradores y tú revisara las nuevas actualizaciones del código para determinar si los cambios estan listos para ser fusionados en la rama 'principal'.

Las pull request solo pueden ser abiertas entre dos ramas que son diferentes.

\*Sin un pull request las actualizaciones podrían tener códigos incorrectos o sin terminar podrían ser agregados\*

#### PROCESO

- Fork del repositorio principal y crear una copia local del mismo
- Hacer cambios localmente
- Hacer push a los cambios hechos localmente al repositorio forkeado
- Realizar pull reques

\*Si no hay cambios por hacer

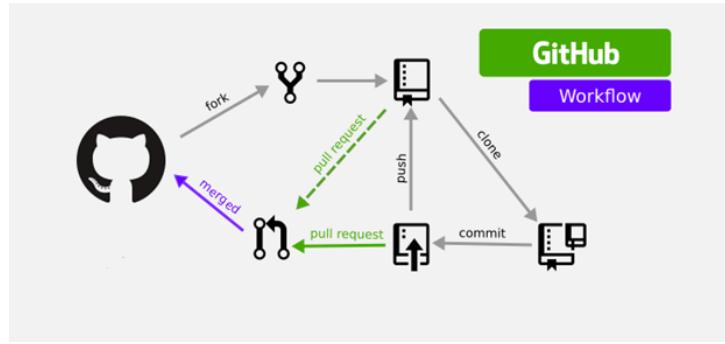
Merge con la rama principal del proyecto.

\*\*\*También se pueden hacer pull request a características o actualizaciones que aún no esten completadas para recibir una posible retroalimentación del equipo.

=> Garantiza que cualquier actualizacion nueva para un proyecto determinado se haya revisado y aprobado minuciosamente antes de fusionarse con el repositorio principal. Previendo problemas futuros y garantizar la experiencia de usuario.

- **fork (bifurcación):**

La función fork se encarga de la creación de una copia de un repositorio en tu cuenta de usuario. Ese repositorio copiado será básicamente un clon del repositorio desde el que se hace el fork, pero a partir de entonces el fork vivirá en un espacio diferente y podrá evolucionar de manera distinta, a tu propio cargo.



"rama externa de un repositorio, colocando esa rama en un nuevo repositorio controlado por otros usuarios"

Generalmente, bifurcar un repositorio nos permite

-experimentar o modificar en el proyecto sin afectar el proyecto original. Las siguientes son las razones para bifurcar el repositorio:

Proponer cambios al proyecto de otra persona.

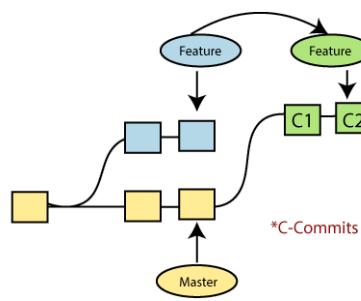
Utilice un proyecto existente como punto de partida.

\*la diferencia significativa es que la bifurcación se usa para crear una copia del lado del servidor y la clonación se usa para crear una copia local del repositorio.

\*\*hacer un fork en Git contribuye a garantizar la seguridad del proyecto original

- **REBASE**

Es el proceso de mover o combinar una secuencia de confirmaciones sobre una nueva confirmación base. Es decir, una forma de mover la totalidad de una rama a otro punto del árbol.



Internamente, Git crea una nueva confirmación y la aplica a la base especificada.

permite el cambio de manera sencilla de un conjunto de commit, realizando modificaciones en el historial del repositorio.

\*git rebase altera el historial de commits

\*\*Solo ejecutar en un repositorio local

Hay dos modos de rebase de Git diferentes,

-Git rebase de modo estándar: toma automáticamente las confirmaciones presentes en su rama de trabajo actual y las aplica inmediatamente al encabezado de la rama pasada.

-Git rebase de modo interactivo: permite cambiar diferentes compromisos en el proceso en lugar de simplemente recoger todo y arrojarlo a la rama pasada. Si usa el modo interactivo, puede eliminar, dividir o alterar las confirmaciones existentes y limpiar el historial.

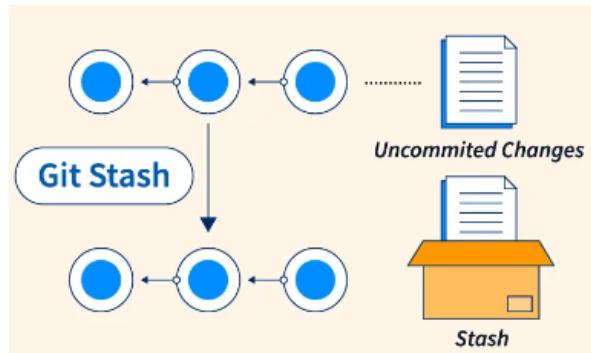
**git rebase <base>** //Realiza el standard rebase

**git rebase – interactive <base>** //Realiza el interactive rebase

- **STASH**

Git stash guarda los cambios no confirmados localmente, lo que le permite realizar cambios, cambiar de rama y realizar otras operaciones de Git.

Es decir, puedes almacenar temporalmente una captura de tus cambios sin enviarlos al repositorio. Está separada del directorio de trabajo (working directory), del área de preparación (staging area), o del repositorio.



\*Utilice git stash cuando desee registrar el estado actual del directorio de trabajo y el índice, pero desee volver a un directorio de trabajo limpio.

//Gurdar cambios en stash

**git stash save "mensaje opcional para ti"**

//Ver cambios guardados en el stash

**git stash list**

//Recuperar cambios en stash

//1.aplica los cambios y deja una copia en el stash

**git stash apply NOMBRE-DEL-STASH**

//2.aplica los cambio y elimina los archivos del stash

**git stash pop NOMBRE-DEL-STASH**

//Borrar cambios guardados en stash

**git stash drop NOMBRE-DEL-STASH**

//Limpiar todo el stash

**git stash clear**

- **CLEAN**

Limpia el árbol de trabajo mediante la eliminación recursiva de archivos que no están bajo control de versiones, a partir del directorio actual.

Si queremos eliminar los archivos no deseados, podemos usar el comando de limpieza en Git.

Por defecto, no eliminará:

los archivos .gitignore

nuevos directorios creados recientemente

archivos de índice.

archivos de confirmación existentes

//Revisar archivos que no tienen seguimiento

**git clean --dry-run**

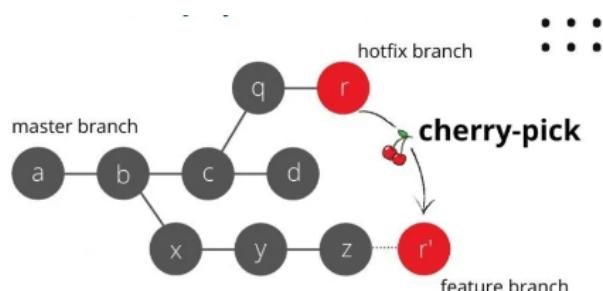
//Eliminar archivos listados de no seguimietno

**git clean /f**

- **CHERRY-PICK**

Es elegir una o más confirmaciones de una rama específica y aplicarla a otra rama.

Permitiendo copiar una confirmación(commit) desde cualquier lugar de su repositorio y agregarla al HEAD de la rama actual.



Ser cuidadoso para evitar inconvenientes como commits duplicados que pueden crear un historial desordenado.

Usarse en caso de

- Cuando se trabaja en funcionalidades diferentes como colaborador, pero en diferentes ramas.

-Reducir bugs

-No perder funcionalidad

**git cherry-pick<commit-hash>**

commit-hash: un hash de confirmación es un identificador único generado por Git. Cada confirmación tiene su único hash de confirmación.

- **Explicar y diagramar en qué consiste MVC**

MVC (Model-View-Controller) es un patrón en el diseño de software utilizado comúnmente para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica comercial y la pantalla del software. Proporcionando una mejor división del trabajo y un mejor mantenimiento.

Las tres partes del patrón de diseño de software MVC se pueden describir de la siguiente manera:

Modelo: Gestiona datos y lógica de negocio.

Define **qué datos debe contener la aplicación**. Si el estado de estos datos cambia, el modelo generalmente notificará a la vista (para que la pantalla pueda cambiar según sea necesario) y, a veces, al controlador (si se necesita una lógica diferente para controlar la vista actualizada).

Vista: maneja el diseño y la visualización

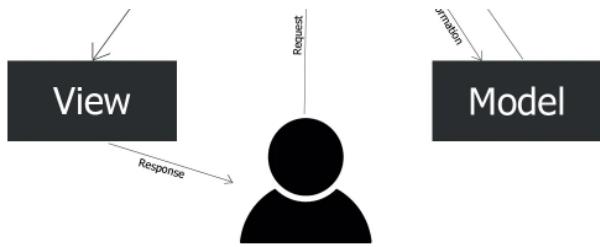
La vista define **cómo se deben mostrar los datos** de la aplicación.

Controlador: enruta los comandos al modelo y las partes de la vista.

El controlador contiene lógica que **actualiza el modelo o la vista en respuesta** a las entradas de los usuarios de la aplicación.

## Model-View-Controller





Usando este patrón, las solicitudes de los usuarios se enrutan a un controlador que es responsable de trabajar con el modelo para realizar acciones de usuario y/o recuperar resultados de consultas. El controlador elige la vista para mostrar al usuario y le proporciona los datos del modelo que requiere.

- **Proyecto personalizado usando JDBC.**

Utilizando como proyecto base web-tracker-student, tomamos el registro de los productos de la tienda, donde podremos visualizar la cantidad de productos con la que contamos, el costo al que compramos nuestros producto y un precio aproximado del producto sumándole el iva.

[LINK](#)

- **Diferencia entre aplicaciones monolítica y microservicios**

**Aplicación Monolítica**

Una aplicación monolítica tiene todas o la mayoría de sus funcionalidades en un único proceso o contenedor.

Todos los componentes de la aplicación (interfaz de usuario, lógica de negocios y capa de acceso a datos) son parte de UNA SOLA UNIDAD - combinadas

- ✓ Todo es desarrollado, desplegado y escalado en una sola unidad.
- ✓ Aplicación deben estar escritas con una sola tecnología
- ✓ Equipos deben ser cuidadosos de no afectar el trabajo de otros.

**Arquitectura monolítica**

The diagram shows a central 'Interfaz usuario (UI)' circle connected to a 'Lógica del negocio' circle, which is then connected to a 'Capa de acceso a los datos' circle. All three components are connected to a single database icon at the bottom.

- Un solo artefacto - Debes de redespelgar la aplicación entera en cada actualización.
- \*Sin modularidad: ie. No permite reutilizar partes de lógica de una aplicación

! Aplicación es más larga y compleja

! Solo se puede escalar la aplicación completa, en lugar de un servicio específico.

! --Mayores costos de infraestructura

! Dificultades si el servicio necesita diferentes versiones de dependencia.

Separaremos la aplicación en pequeños e independientes servicios/componentes basados en las funcionalidades comerciales y no en las funcionalidades técnicas.

✓ Componente/Servicio debe ser autónomo e independiente entre sí

✓ Desarrollarse, implementarse y escalarse por separado. (PÉRDIDA DE ACOMPLAMIENTO)

**Arquitectura • microservicios**

The diagram shows a central 'Interfaz usuario (UI)' circle connected to five separate 'Microservice' circles. Each 'Microservice' circle is also connected to a database icon below it.

**COMUNICACIÓN ENTRE SERVICIOS**

- ⚙ API (Application Programming Interface) = Envío de solicitudes a la API -- COMUNICACIÓN SÍNCRONA (Espera por la respuesta)
- ⚙ MessageBroker = Primero se envían mensajes al intermediario y se reenvía al respectivo servicio -- COMUNICACIÓN ASÍNCRONA

# Microservicios



Liberación del proceso lleva más tiempo.

## Service Mesh (Malla de servicio)

\*Común en Kubernetes. Teniendo una especie de servicio de ayuda que se hace cargo de la lógica de comunicación completa, por lo que no codificas tú.



Configurar la comunicación entre los servicios



Difícil mantener una visión general cuando un microservicio tiene algún problema.

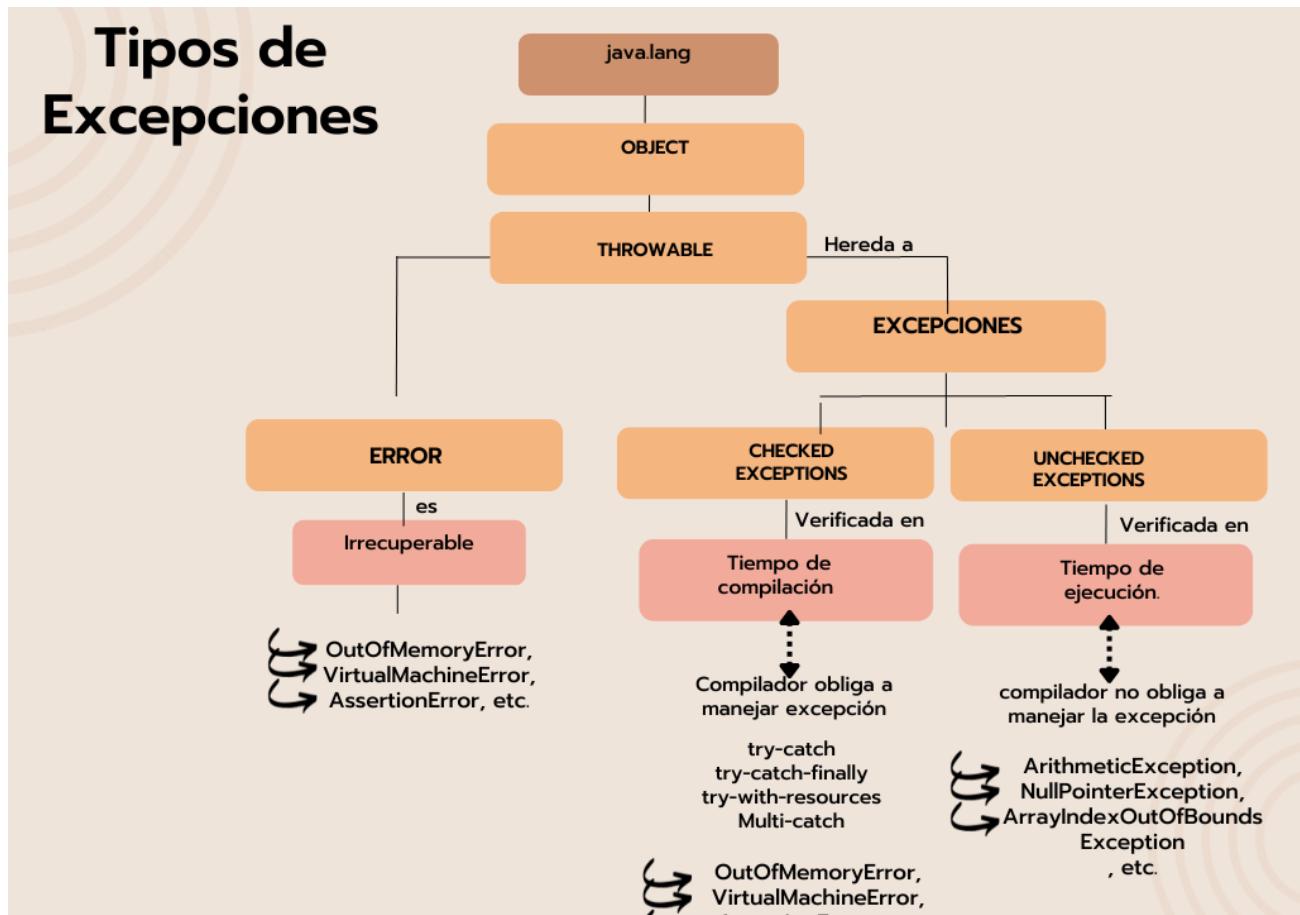
- Explicar y diagramar los tres tipos de excepciones en Java.

Definición: Una 'excepción' es un evento, que ocurre durante la ejecución de un programa, que interrumpe el flujo normal de las instrucciones del programa.

Cuando ocurre un error dentro de un método, el método crea un objeto y lo pasa al sistema de 'tiempo de ejecución'. El objeto, llamado 'objeto de excepción', contiene información sobre el error, incluido su tipo y el estado del programa cuando ocurrió el error. Crear un objeto de 'excepción' y entregarlo al sistema de tiempo de ejecución se denomina 'lanzar' una excepción.

Existen principalmente dos tipos de excepciones: Checked y unchecked.

Un error se considera como la excepción no verificada. Sin embargo, según Oracle, existen tres tipos de excepciones, a saber:





- **Explicar try-with-resource y multicatch.**

- TRY-WITH-RESOURCES (incorporada en Java7): Es un try statement que declara una o más recursos.

Cada recurso debe ser un objeto de una clase que implemente a la interfaz AutoCloseable y por ende, proporciona un método llamado close.

La intruccion try-with-resources llama de manera implícita al método close del objeto al final del bloque try, asegurándose que cada recurso ha sido cerrado al final del statment.

\*Se pueden asignar varios recursos en los paréntesis que van después del try, separándolos por punto y coma (;).

Una declaración try-with-resources puede tener bloques catch y finally como una declaración de try normal. En una sentencia try-with-resources, cualquier bloque catch o finally se ejecuta después de que se hayan cerrado los recursos declarados.

- MULTI-CATCH = Catching Multiple Exceptions (incorporado en Java 7)

Captura más de un tipo de excepción en un solo bloque catch.

La cláusula catch especifica los tipos de excepciones que puede manejar el bloque, y cada tipo de excepción está separado por una barra vertical (|).

\*Nota: si un bloque (catch) maneja múltiples excepciones, el parámetro catch es implícitamente (final). Esto significa que no podemos asignar ningún valor a los parámetros de captura.

- **Proyecto con pruebas unitarias**

Calcula el valor nutrimental de las comidas, tomando en cuenta que existe al menos un ingrediente y mostrando el total valor nutrimental de la comida creada.

Formar grupos de alumno y obtener promedios de edades y calificaciones, según sean agrupados.

Fuentes:

What is a Pull Request? (2021, 28 julio). PagerDuty. <https://www.pagerduty.com/resources/learn/what-is-a-pull-request/>

