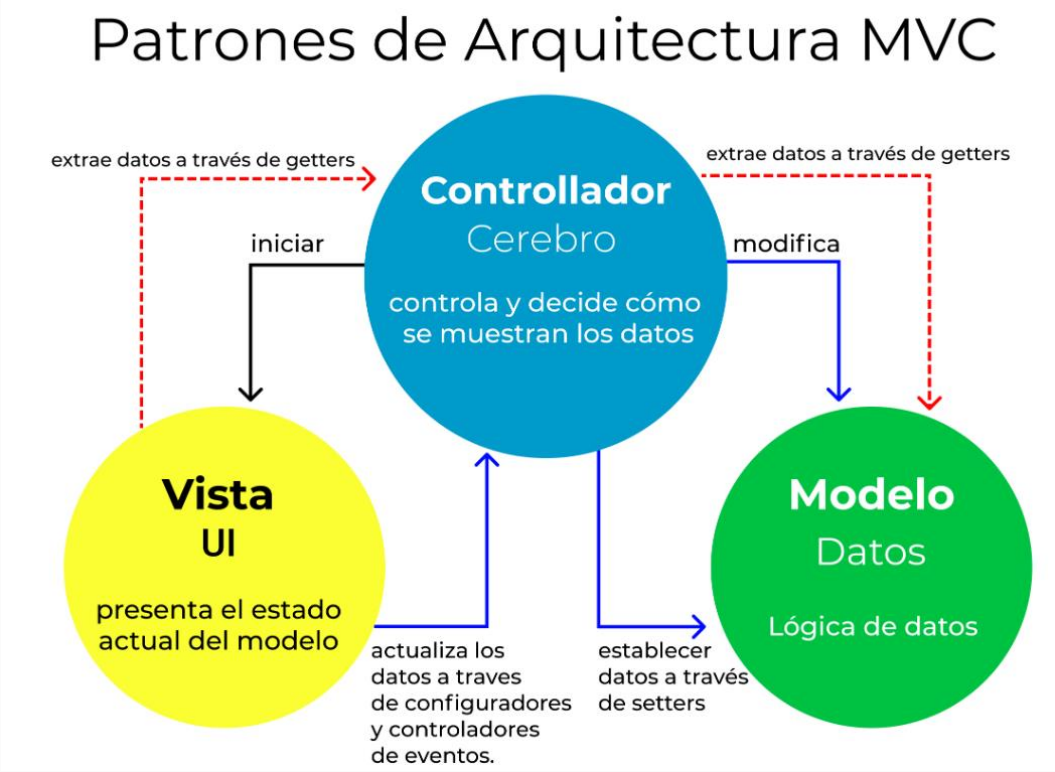
# MVC

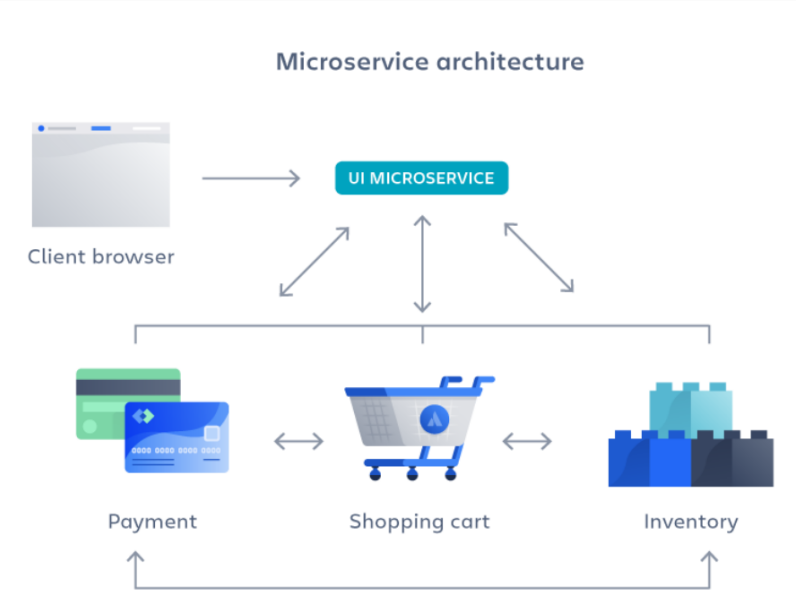



**MVC** (Modelo-Vista-Controlador) es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica de negocios y su visualización. Esta "separación de preocupaciones" proporciona una mejor división del trabajo y una mejora de mantenimiento.

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

1. Modelo: Maneja datos y lógica de negocios.
2. Vista: Se encarga del diseño y presentación (frontend o interfaz gráfica de usuario (GUI)
3. Controlador: Enruta comandos a los modelos y vistas (El cerebro de la aplicación que controla como se muestran los datos.)

# MICROSERVICIOS




Una arquitectura de microservicios, o simplemente "microservicios", es un método de arquitectura que se basa en una serie de servicios que se pueden implementar de forma independiente. Estos servicios tienen su propia lógica empresarial y base de datos con un objetivo específico. La actualización, las pruebas, la implementación y el escalado se llevan a cabo dentro de cada servicio.

Los microservicios desacoplan los principales intereses específicos de dominios empresariales en bases de código independientes. Los microservicios no reducen la complejidad, pero hacen que cualquier complejidad sea visible y más gestionable, ya que separan las tareas en procesos más pequeños que funcionan de manera independiente entre sí y contribuyen al conjunto global.

Una arquitectura monolítica es un modelo tradicional de un programa de software que se compila como una unidad unificada y que es autónoma e independiente de otras aplicaciones. La palabra "monolito" suele evocar algo grande y glacial, una imagen que no está alejada de la realidad de las arquitecturas monolíticas para el diseño de software. Una arquitectura monolítica es una red informática grande y única, con una base de código que aúna todos los intereses empresariales. Para hacer cambios en este tipo de aplicación, hay actualizar toda la pila, lo que requiere acceder a la base de código y compilar e implementar una versión actualizada de la interfaz del lado del servicio; esto hace que las actualizaciones sean restrictivas y lentas.

# Excepciones

Las excepciones son situaciones anómalas que pueden ocurrir durante la ejecución de las aplicaciones, como, por ejemplo, acceder a una posición de un vector fuera rango.

En algunos casos las excepciones no se podrán gestionar, por ejemplo, los de la propia JVM, y en otros casos sí.

Java proporciona un mecanismo de gestión de excepciones en los casos en los que estemos obligados a gestionarlas. En caso de que no queramos gestionar algún tipo de excepción estaremos obligados a indicarlo explícitamente. No podremos compilar el código a menos que gestionemos la excepción, o indiquemos explícitamente que no lo queremos hacer.

La definición de excepciones propias se hace a través del mecanismo de extensión de clases, la Herencia

| | |
|---|---|
| **Excepciones comprobadas** | Se conocen como excepciones de tiempo en compilación, ya que estas excepciones son comprobadas por el compilador durante el proceso de compilación para confirmar si el programador maneja la excepción o no. Si no es así muestra un error de compilación. |
| **Excepciones en tiempo de ejecución** | Se produce simplemente porque el programador ha cometido un error, has escrito el código, todo le parece bien al compilador y cuando se va a ejecutar, falla porque intentó acceder a un elemento que no existe o un error lógico hizo que se llamara a un método con un valor nulo. |
| **Errores** | Están fuera de nuestro alcance por lo tanto no podemos meterlos en tratamiento para corregirlos. |

# Multicatch

# Y try-with-resources

**Multi-catch**

Concepto para manejo de excepciones, es cuando dentro de un único catch se puede manejar varias excepciones asumiendo todas las excepciones que se mete al catch que tengan el mismo comportamiento.

Un bloque try puede ir seguido de uno o más bloques catch. Cada bloque catch debe contener un controlador de excepción diferente. Por lo tanto, si tiene que realizar diferentes tareas cuando ocurren diferentes excepciones, use el bloque de captura múltiple de Java.

- A la vez solo ocurre una excepción y a la vez solo se ejecuta un bloque catch.
- Todos los bloques catch deben ordenarse del más específico al más general, es decir, catch para ArithmeticException debe ir antes que catch para Exception.

**Try-with-resources**

- Sirve para declarar recursos, un recurso es un objeto que debe cerrarse una vez que ya halla sido utilizado en el programa.
- Asegura que cada objeto que implemente la clase AutoClaseable se cierre de manera automática para no seguir ocupando espacio en la memoria.

Si ocurre una excepción en la instrucción particular del bloque try, el resto del código del bloque no se ejecutará. Por lo tanto, se recomienda no mantener el código en el bloque de prueba que no generará una excepción.

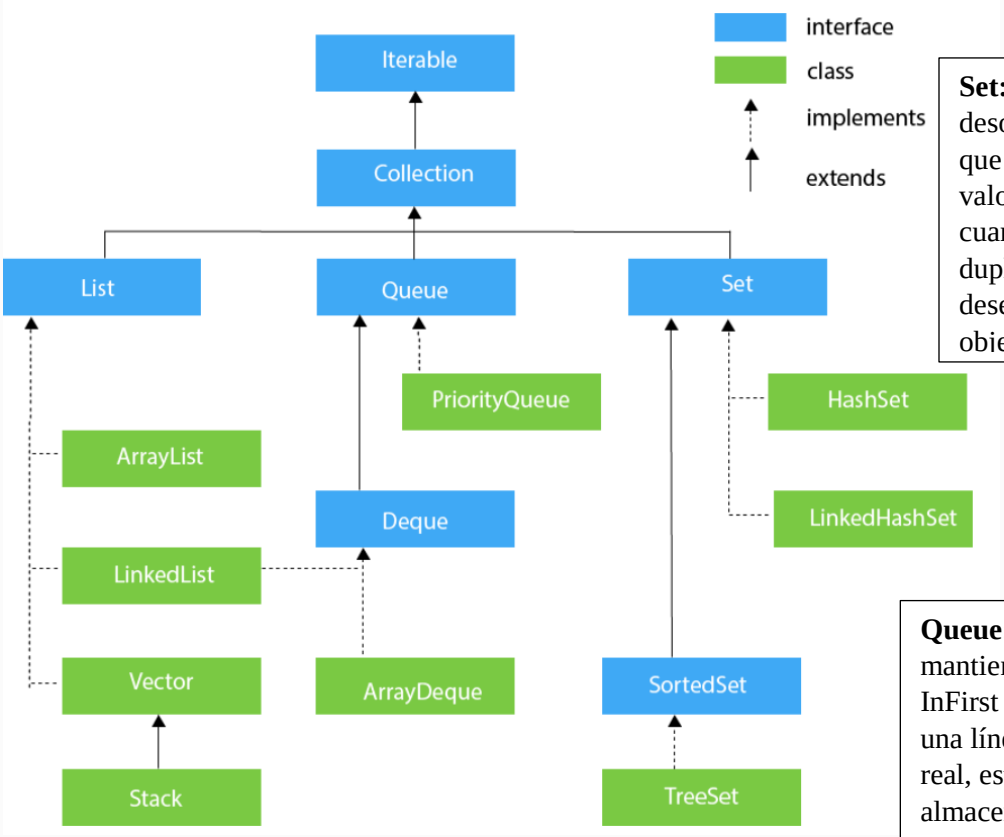
El bloque de prueba de Java debe ir seguido de un bloque de catch o finally block.

# Collections

La **Colección en Java** es un marco que proporciona una arquitectura para almacenar y manipular el grupo de objetos.

Las colecciones de Java pueden lograr todas las operaciones que realiza en datos, como búsqueda, clasificación, inserción, manipulación y eliminación.

Colección Java significa una sola unidad de objetos. El marco de Java Collection proporciona muchas **interfaces** (Set, List, Queue, Deque) y **clases** ( ArrayList , Vector, LinkedList , PriorityQueue , HashSet, LinkedHashSet, TreeSet).




**Set:** es una colección desordenada de objetos en la que no se pueden almacenar valores duplicados, se utiliza cuando queremos evitar la duplicación de objetos y deseamos almacenar solo obietos únicos.

**Queue:** la interfaz de cola mantiene el orden FIFO (First InFirst Out) de forma similar a una línea de cola del mundo real, esta interfaz se dedica a almacenar todos los elementos cuyo orden es importante.

**Map**: una interfaz, es una estructura de datos que soporta el par clave-valor para mapear los datos. Esta interfaz no soporta claves duplicadas porque una misma clave no puede múltiples mapeos, sin embrago permite valores duplicados en diferentes claves.
Es útil si hay datos y queremos realizar operaciones en base a la clave.