Algunos Conceptos

MADE BY: OSCAR ALEXIS RAMIREZ

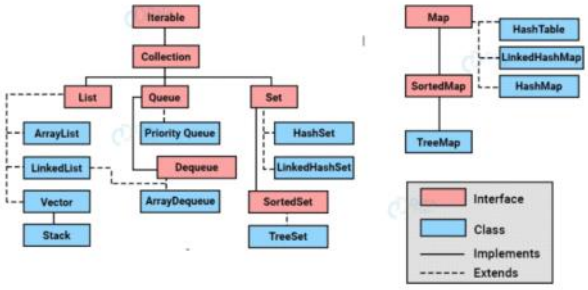
ARQUITECTURA MONOLOTICA VS ARQUITECTURA MICROSERVICIOS

Arquitectura Monolítica	Arquitectura microservicios
Son aplicaciones que se compilan como una sola unidad unificada.	Son servicios pequeños que se pueden implementar de forma independiente.
Son de Implementación sencilla es decir un único archivo o directorio ejecutable facilita la implementación.	Tienen su propia lógica empresarial y base de datos con un objetivo específico
Desarrollo a nivel de código sencillo	Son agiles porque promueve formas ágiles de trabajar con equipos pequeños que implementen con frecuencia.
Facilidad para realizar pruebas	Es de fácil escalabilidad ya que cuando se está alcanzando su capacidad de carga se pueden implementar rápidamente nuevas instancias de ese servicio al clúster que lo acompaña para aliviar la presión.
Facilidad para realizar depuraciones	Muy fácil de mantener y probar: los equipos pueden hacer pruebas con el código y dar marcha atrás si algo no funciona como esperan.
Desarrollo es más complejo y lento	Fiabilidad ya que puedes implementar cambios para un servicio en concreto sin el riesgo de que se caiga toda la aplicación.
No se pueden escalar componentes individuales.	Costes exponenciales de infraestructura
Si hay un error en algún módulo, puede afectar a la disponibilidad de toda la aplicación.	Desafíos para la depuración porque cada microservicio tiene su propio conjunto de registros, lo que complica la depuración.
Dificultad para ser mantenible	Falta de estandarización ya que sin una plataforma común, puede haber una proliferación de idiomas, de estándares de registro y de supervisión.

TIPO DE COLLECCIONES

Collection

1. Es la interface raíz de un conjunto de colecciones.
2. Una colección representa un grupo de objetos, conocidos como sus elementos .
3. El JDK no proporciona implementaciones directas de esta interfaz: la implementación se dividió en sus interfaces secundarias y sus clases de implementación.
4. La interfaz de colección proporciona los métodos generales más comunes que se aplican a cualquier objeto de colección (implementaciones de Lista, Conjunto, Cola).



List Interface

1. Una colección ordenada (también conocida como secuencia).
2. Se conserva el orden de inserción, si el usuario usó la implementación de tipo Lista, el usuario tiene control sobre el orden, porque el orden de inserción y el orden de recuperación de un elemento son los mismos.
3. Si las implementaciones de tipo List usaron elementos duplicados permitidos para insertar.

SET Interface

1. El orden de inserción no se conserva, si el usuario usó la implementación de tipo Set, el usuario no tiene control sobre el orden, porque el orden de inserción y el orden de recuperación de un elemento no son los mismos.
2. Si se utilizan implementaciones de tipo Set No se permite insertar elementos duplicados.

Queue Interface

1. Colección diseñada para retener elementos antes de su procesamiento.
2. Una cola admite las operaciones de inserción y eliminación mediante el método FIFO (primero en entrar, primero en salir).
3. La cola proporciona métodos de procesamiento previo.

Map Interface

1. Objeto que asigna claves a valores.
2. Un mapa no puede contener claves duplicadas; cada clave puede corresponder a un valor como máximo.
3. Map representa cada elemento como un par clave-valor, y ese par clave-valor se denomina Entrada.
4. La interfaz de Map proporciona los métodos más comunes aplicables a cualquier objeto de tipo Map.

EXCEPCIONES

Una excepción, es el resultado de realizar alguna operación cuyo resultado puede causar un error que rompa con el flujo de ejecución de una parte de nuestro código.
En la gran mayoría de casos Java no gestiona las excepciones necesarias, uno debe indicarlas, ya que son de acuerdo a la necesidad.
Para lo cual se clasifican en 3 tipos.

1. **Excepciones Comprobadas**
   Son excepciones de tiempo de compilación ya que estás son comprobadas en tiempo de compilación con el fin de comprobar si es que fue implementada o no. Por ejemplo cuando trabajamos con archivos es necesario implementar excepciones.

2. **Excepciones en tiempo de ejecución**
   Son errores que no son perceptibles al programar, sin embargo pueden suceder al momento de ejecutarse el programa. Por ejemplo una división entre cero.

3. **Errores**
   Son aquellos que están fuera de nuestro alcance como programadores, por lo cual su implementación resulta difícil o imposible para solucionarlos.

MULTICATCH Y TRY-WITH-RESOURCES

1. **Multicatching**
   El concepto de Java MultiCatch Block es uno de los conceptos más clásicos de Manejo de Excepciones. En un bloque de código podemos tener la necesidad de Capturar varias Excepciones problemáticas.
   Por ejemplo al leer un numero para una división una excepción podría ser que el número no sea cero otra para validar que el número sea un int no una cadena. Etc.

```
try {
      fw = new FileWriter("nuevo.txt");
      fw.write("hola soy un fichero de texto");
      DriverManager.getConnection("jdbc:mysql://localhost:3306", "root", "root");
      System.out.println("fichero creado");
      System.out.println("acceso a base de datos");
    } catch (IOException e1) {
      System.out.println("error de fichero"+e1);
    } catch (SQLException e1) {
      System.out.println("error de SQL"+e1);
    }
```

   En el siguiente ejemplo se maneja doble catch para prevenir ciertas excepciones que puedan ocurrir.

2. **Excepciones en tiempo de ejecución**
   En Java, la declaración de prueba con recursos es una declaración de prueba que declara uno o más recursos. El recurso es como un objeto que debe cerrarse después de finalizar el programa. La instrucción try-with-resources garantiza que cada recurso se cierre al final de la ejecución de la instrucción.
   En la imagen se aprecia el siguiente ejemplo.

```
import java.io.FileOutputStream;
public class TryWithResources {
public static void main(String args[]){
        // Using try-with-resources
        try(FileOutputStream fileOutputStream =newFileOutputStream("tbc.txt")){
            String msg = "Welcome to javaTpoint!";
            byte byteArray[] = msg.getBytes();
            fileOutputStream.write(byteArray);
            System.out.println("Message written to file successfuly!");
        }catch(Exception exception){
            System.out.println(exception);
        }
    }
}
```