**20/08/25 – BACKEND DE APLICACIONES**

**ARREGLOS EN JAVA**

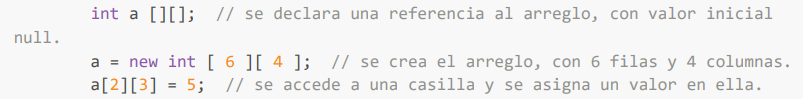
Son vectores, arreglos unidimensionales. Se trata de una colección de valores que deben ser del mismo tipo y que se organiza de forma en que cada componente individual es identificado por un índice (que permiten el acceso y uso de cada componente).

Para declarar un arreglo se debe colocar el tipo base del arreglo, el nombre del arreglo y unos corchetes vacíos: int[] nombre = new int[tamaño del arreglo].

Todo objeto arreglo en Java provee un atributo llamado length, que contiene el tamaño del arreglo tal como fue declarado al crear ese arreglo con new.

**MATRICES EN JAVA**

Son arreglos bidimensionales, cuyos elementos están dispuestos en forma de tabla, con varias filas y columnas. Para entrar a un componente, debe darse el índice de la fila de este y también el índice de la columna. Como los índices requeridos son dos, el arreglo es de dimensión dos.



Diagrama

El contenido generado por IA puede ser incorrecto.Manejo de Excepciones en Java: un error en tiempo de ejecución provocado por circunstancias anormales (errores matemáticos, índices fuera de rango, casting imposible de realizar, puntero nulo invocando a un método, etc.) se representa como un objeto. Esos objetos que representan errores de ejecución se llaman objetos de excepción o simplemente excepciones y forman parte de una jerarquía de clases cuya clase base es la clase Throwable.

Los errores representados por la clase Error son errores graves de hardware o de sistema frente a los cuales no se espera que el programador pueda hacer nada más que darse por notificado del hecho.

Los errores representados por la clase Exception son errores comunes de programación, algunos más graves que otros, que pueden requerir que el programador se vea obligado a escribir código de respuesta para esas excepciones, pues de lo contrario el programa no compilará.

En Java, cuando un segmento de código puede lanzar una excepción del tipo checked, el compilador obliga al programador a contemplar esa posibilidad. Esto significa que debe escribirse código para manejar dicha excepción, aun cuando en la práctica nunca llegue a producirse. Generalmente, todas las clases de excepción que heredan de la clase Exception son chequeadas, excepto aquellas que derivan de RuntimeException. Un ejemplo claro son las excepciones de entrada y salida (IO), que provienen de la clase IOException. Estas son chequeadas y, por lo tanto, es obligatorio incluir un bloque de tratamiento para manejarlas adecuadamente.

En contraste, las excepciones no chequeadas no requieren que el programador escriba código de respuesta. La decisión de tratarlas o no queda a su criterio. Todas las clases que heredan de RuntimeException, así como la clase Error, entran en esta categoría. Si el programador no las maneja y llegan a producirse, el programa o el método se interrumpe mostrando un mensaje de error correspondiente, pero no se presentan problemas de compilación. Entre las más comunes derivadas de RuntimeException están NullPointerException, ArithmeticException, IndexOutBoundsException y NegativeArraySizeException. Por parte de IOException, las más habituales son EOFException, FileNotFoundException, InterruptedIOException y ObjectStreamException.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.Finalmente, existen dos formas principales de responder a una excepción en el código. En el caso de las excepciones chequeadas, el programador está obligado a escoger una de estas alternativas. En el caso de las no chequeadas, puede optar por no hacer nada y dejar que el error interrumpa el programa, o bien tratarlas de la misma manera que una excepción chequeada, ya sea gestionando su ocurrencia o informando de su posible existencia.

**TRATAMIENTO DE EXCEPCIONES**: consiste en capturarla con un bloque try – catch, en dicho bloque se comprueba si una excepción se ha producido, y actuar en consecuencia. Para ello se utilizan las palabras reservadas try, catch y finally. El bloque try tiene que ir seguido, al menos, por una cláusula catch o una cláusula finally. La sintaxis general del bloque try consiste en la palabra clave try y una o más sentencias entre llaves.

Cuando se colocan varios controladores de excepción, es decir, varias sentencias catch, el orden en que aparecen en el programa es importante, especialmente si alguno de los controladores engloba a otros en el árbol de jerarquía. Se deben colocar primero los controladores que manejen las excepciones más alejadas en el árbol de jerarquía. Por último el bloque finally, es el bloque de código que se ejecuta siempre, haya o no excepción (trozo de código que se ejecuta independientemente de lo que se haga en el bloque try).

El objeto generado por la JVM para representar la excepción dispone de una serie de métodos que permiten que el programador tenga mayor conocimiento del error producido. Uno de esos métodos es getMessage() (heredado desde Throwable) que retorna un String con una descripción del error que provocó la excepción. El mismo puede usarse para mostrar un mensaje más claro.

**Try with resources (Try con recursos)**

Se trata de un bloque try en el cual se declaran en forma especial ciertos recursos. Esos recursos son objetos que deben ser cerrados al terminar el programa (como en el caso del ejemplo que hemos mostrado hasta aquí). El bloque try with resources garantiza que esos recursos serán efectivamente cerrados (en forma automática) al terminar de ejecutarse el bloque try. Técnicamente, los recursos declarados en un try with resources son objetos de cualquier clase que implemente la interface java.lang.AutoCloseable o su derivada java.lang.Closeable y el único método que estas declaran es justamente el método close().

**Throws (avisando posible ocurrencia de excepciones)**

throws se utiliza para identificar la lista posible de excepciones que un método puede lanzar. Si un método es capaz de provocar una excepción que no maneja él mismo, debería especificar este comportamiento, para que todos los métodos que lo llamen puedan colocar protecciones frente a esa excepción. Para la mayoría de las subclases de la clase Exception, el compilador Java obliga a declarar qué tipos podrá lanzar un método. Si el tipo de excepción es Error o RuntimeException, o cualquiera de sus subclases, no se aplica esta regla, dado que no se espera que se produzcan como resultado del funcionamiento normal del programa. tipo nombreMetodo( argumentos ) throws excepciones { }.



La sentencia throw se utiliza para lanzar explícitamente una excepción. En primer lugar se debe obtener un descriptor de un objeto Throwable, bien mediante un parámetro en una cláusula catch o, se puede crear utilizando el operador new. La forma general de la sentencia throw es: throw objetoThrowable;

**TESTING**: serie de pruebas, donde el programador ejecuta una parte del programa, conociendo de antemano los resultados que deberían darse y verifica que efectivamente se estén entregando esos resultados. Estas pruebas de partes del programa se conocen como pruebas unitarias, o tests unitarios. El ecosistema Java posee varias herramientas pensadas para realizar este tipo de tests, y la mas común es la utilización de una librería llamada JUnit.

Particularmente la versión 5 de JUnit se compone de tres componentes:

* JUnit Platform: Que es la plataforma que permite el descubrimiento y ejecución de las pruebas unitarias. Esta plataforma se encuentra integrada en casi todos los IDE de java y en las herramientas de construcción como Maven o Gradle
* JUnit Jupiter: Es un modelo de programación que nos permite escribir los tests y provee una serie de anotaciones y extensiones que ayudan a la escritura de los tests.
* JUnit Vintage: Es una capa de compatibilidad para poder seguir usando tests escritos para JUnit4 en proyectos que usan JUnit 5.

Texto

El contenido generado por IA puede ser incorrecto.Texto, Carta

El contenido generado por IA puede ser incorrecto.Para poder utilizar JUnit en un proyecto maven, basta con agregar una dependencia al artefacto con groupId org.junit.jupiter y artifactId junit.jupiter. En general al referirse a una dependencia de maven se suele usar la forma groupId:artifactId, o sea para este caso la dependencia necesaria sería org.junit.jupiter:junit.jupiter Dicha dependencia se agrega en la sección del archivo pom.xml de la siguiente forma:

Por convención de Maven, dentro de la carpeta src hay dos carpetas, main y test. La carpeta main es donde van a estar todos los archivos de código fuente que forman parte del código de producción, o sea el que se va a terminar empaquetando con el proyecto. La carpeta test está para que en ella se ubiquen todos los test unitarios, estos fuentes sólo se compilan durante la ejecución de los tests, pero no forman parte del producto. Dentro de JUnit un unit test es básicamente un método anotado con la anotación @Test, y los tests se agrupan en Suites, que son clases cuyo nombre generalmente termina en Test y contienen uno o más métodos anotados con la anotación @Test.

Además de @Test hay varias anotaciones más que se pueden usar para marcar métodos, algunas de ellas son:

* @BeforeEach: Marca un método que se va a ejecutar antes de la ejecución de cada uno de los tests de la suite
* @BeforeAll: Marca un método que se va a ejecutar antes de la ejecución de todos los tests de la suite (sólo se ejecuta una vez por suite)
* @AfterEach: Análogo a BeforeEach, pero se ejecuta luego de cada test
* @AfterAll: Igualmente análogo a BeforeAll, sólo se va a ejecutar una vez al terminar de ejecutar todos los tests de la suite
* @Disabled: Permite desactivar un test, para que no se ejecute
* @Timeout: Permite establecer un tiempo máximo de ejecución para un test, si este tiempo se excede el test falla
* Interfaz de usuario gráfica, Texto, Aplicación

  El contenido generado por IA puede ser incorrecto.@ParameterizedTest: Esta técnica permite testear múltiples valores dinámicamente y validar comportamientos agregados.

Comprobaciones: se espera que además de ejecutar algo y ver que no se produzcan excepciones, se compruebe que los resultados provistos por el código sean los correctos. Esto se hace mediante comprobaciones, conocidas como assertions. Si la comprobación es correcta, sigue la ejecución, pero si no es correcta se interrumpe la ejecución del test con un error.

* assertTrue(boolean)\*: Comprueba que el boolean sea true, o falla
* assertTrue(boolean, mensaje): Comprueba que el boolean sea true o falla informando el mensaje indicado
* assertFalse(boolean): Comprueba que el boolean sea false, o falla
* assertFalse(boolean, mensaje): Comprueba que el boolean sea false o falla informando el mensaje indicado
* assertEquals(esperado, valor): Comprueba que valor sea igual a esperado o falla. Este método tiene muchas variantes con diferentes tipos, notable de destacar es la variante de float que admite un tercer parámetro indicando un delta que tiene que superarse para que se considere que los valores no son iguales.
* assertEquals(esperado, valor, mensaje): Igual que el anterior, pero agregando el mensaje a mostrar en caso de falla.
* assertNotEquals(esperado, valor): Contrario a assertEquals
* assertNotEquals(esperado, valor, mensaje): Contrario a assertEquals
* fail()\*\*: hace fallar el test
* fail(mensaje)\*\*: hace fallar el test informando un mensaje
* assertThows(clase, ejecutable): Comprueba que el código ejecutado en ejecutable, que es una interfaz funcional donde se puede usar un lambda, lance una Exception del tipo clase
* assertDoesNotThrow(ejecutable): Comprueba que el código ejecutado en ejecutable NO lance una exception.
* assertAll: permite validar todos los atributos relevantes sin detener la ejecución ante el primer fallo.

Todos los métodos definidos en Assertions son estáticos, con lo cual se pueden usar de la forma Assertions.assertEquals(a, b), o también se suele hacer de forma habitual un import static para que todos los métodos assert estén disponibles para llamar directamente.



Mocking: es normal que en algún punto tengamos que hacer que algún objeto devuelva un valor predefinido, esperado por el test. Esto se logra implementando una versión del repositorio que devuelve valores fijos, o configurables, y usando dicha versión al instanciar el servicio en el test.