

1. ¿Qué son?

En Java cuando existe la clase Error, los objetos asociados a esta clase son los encargados de señalar al programador o usuario que hubo un error al momento de compilar el archivo Java o al momento de ejecución del programa. Dicha clase contiene una jerarquía de Errores.

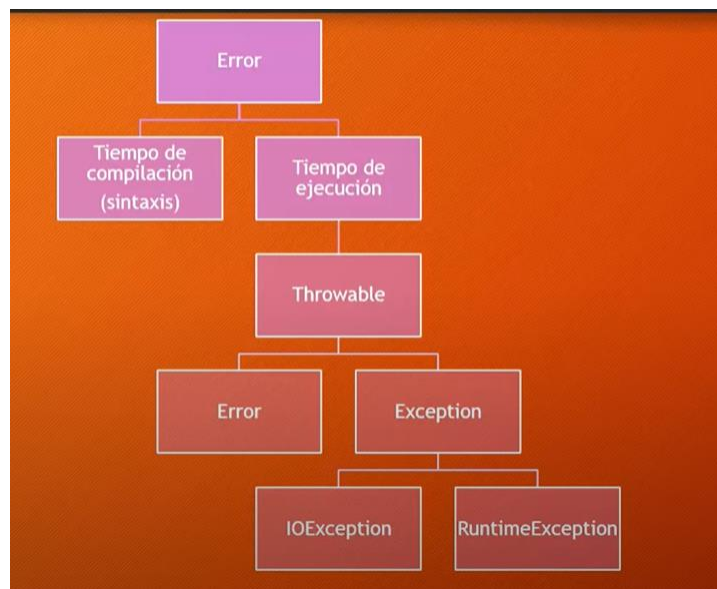


Ilustración 1: Jerarquía de Errores en Java.

Una subclase de tipo Error, es la clase Exception dicha clase almacena los errores comunes de Java que se pueden detectar, por ejemplo: almacenar elemento String en un int, dividir entre 0, entre otros.

2. ¿Para qué sirven?

Las excepciones tienen como propósito controlar las posibles situaciones que pueden pasar en un programa. En caso de toparse con estos posibles comportamientos el programador puede indicar como el código debe responder y solucionarlo o indicarlo. Las excepciones no van a solucionar directamente el problema, pero si sirven para alertarnos o realizar un comportamiento establecido por el programador.

3. ¿Cómo se hace el manejo de errores en lenguajes sin excepciones?

En los lenguajes que no tienen excepciones es más difícil el tratamiento de los errores. Ya que, para tratar dichos errores, debes conocer antemano el porqué, del mal funcionamiento. Con el fin de poner las condiciones necesarias, y así no interrumpir la ejecución del programa.

```
int foo()
{
    int* p;

    malloc(p, sizeof(int));
    if (p == NULL)
    { /* hubo algún error */
        return 1;
    }
    ...
    return 0;
}

int main()
{
    if (foo())
    { /* reportar que hubo un error */
        printf("La función 'foo' falló.");
        return EXIT_FAIL;
    }

    return EXIT_SUCCESS;
}
```

```
try
{
    ...sentencia.1...
    ...sentencia.2...
    .....
    ...sentencia.n...
}
catch ([argumentos])
{
}
```

Ilustración 2: Manejo de Excepciones de C vs Java

4. ¿Cuáles son buenas prácticas a la hora de utilizar excepciones?

Algunas buenas prácticas al momento de programar las excepciones son:

- Limpiar los recursos en el bloque `Finally` o utilizar la sentencia de `Try` con los recursos:
- Por ejemplo, al momento de abrir un archivo, este puede llegar a tirar una posible excepción, por lo que se recomienda el uso del `Finally`, para asegurar que el archivo abierto no consuma más recursos.
- Utilizar Excepciones específicas: Cuando existe una jerarquía para el tipo de excepciones como en Java, se recomienda el uso de las subclases, en vez de la clase padre, con el fin que no se pierda información.
- Documente las excepciones que especifique: en un método cuando tira una excepción esencial documentarla en Javadoc, con el fin que otros programadores conozcan su existencia.
- Captura la excepción más específica primero: sino las demás excepciones se perderán, ya que la clase padre engloba a las demás.
- No ignore excepciones: debido a cuán más extenso sea el programa, será más difícil la búsqueda de errores. Por lo que se recomienda desde el principio considerarlas, con el fin de facilitar la depuración.
- No registre una excepción para luego lanzar: es importante enseñar la información, pero no saturar de información. Se debe seleccionar los mensajes necesarios para almacenar la información o mostrar en pantalla la información.

5. ¿Cómo puede crear sus propias excepciones?

Se crea una clase que extiende se extiende de Exception, o algunas de sus clases hijas (RuntimeException o IOException) (Ilustración 3), y luego en otro método en él ocupa agregar la información de la excepción haces que tire una excepción del nuevo tipo (Ilustración 4).

```
package main;

public class CustomException extends Exception{

    public static final long serialVersionUID = 700L;

    public CustomException(String mensaje){
        super(mensaje);
    }

}
```

Ilustración 3: Creación de Nueva Excepción

```
package exception;

public class Handler {
    Handler()
    {

    }

    public void funcion (Object object) throws CustomException {

        boolean condition = false, condition2 = false;
        //Logic for programan
        if(condition){
            throw new CustomException("No funciona por ...");
        }else if(condition2){
            throw new CustomException("No funciona por ...");
        }else {
            throw new CustomException("No funciona por ...");
        }
    }


}
```

Ilustración 4: Métodos que utiliza la excepción

```
package exception;

public class Main {
    public static void main(String[] args) {

        try {
            Handler.funcion("ARGUMENTOS");
        } catch (CustomException e) {
            e.printStackTrace();
        }
    }
}
```



```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" ...
exception.CustomException Create breakpoint : No funciona por ...
    at exception.Handler.funcion(Handler.java:18)
    at exception.Main.main(Main.java:7)

Process finished with exit code 0
```

Ilustración 5: Funcionamiento de la Excepción

Por último, cuando hagas referencia a la función tendrás que rodearlo en un try catch, con el fin de ejecutar, la excepción (Ilustración 5).

6. Investigar sobre *frameworks* para logs. En el ecosistema Java, existen muchas implementaciones de frameworks para errores. Se recomienda investigar cuáles son los más populares y seleccionar uno. Posteriormente, para el *framework* seleccionado, explicar cuáles son sus principales características.

Características del *framework* Log4J:

- Su funcionamiento se basa en tres elementos básico: *loggers*, *appenders* y *layouts*.
- Los *loggers* se asocian a nombres de paquetes o clases, estas recogen los mensajes de tipo log que lanzan estos paquetes o clases.
- Cuando se registra el mensaje de log, este se debe mostrar en alguna parte. Para ello se envía al objeto tipo *appender*.
- Para dar formato a estos mensajes de log se utiliza el *layout*.

Referencias

- Curso Java*. You Tube. (2014). Recuperado 20 October 2020, de https://www.youtube.com/watch?v=coK4jM5wvko&list=PLU8oAIHdN5BktAXdEVCLUYzvDyqRQJ2lk&ab_channel=pildorasinformaticas.
- Java Platform SE 7*. Docs.oracle.com. Retrieved 20 October 2020, de <https://docs.oracle.com/javase/7/docs/api/>.
- Gomez, F. Excepciones - Fernando A. Gómez F.. Sites.google.com. Recuperado 20 October 2020, de <https://sites.google.com/site/fernandoagomezf/programacion-en-c/tips-de-programador-c/iso-c/excepciones>.
- Briceño, G. (2017). Java: Buenas prácticas para el manejo de Excepciones. Club de Tecnología. Recuperado 20 Octubre 2020, de <https://www.clubdetecnologia.net/blog/2017/java-buenas-practicas-para-el-manejo-de-excepciones/>.
- Curso Java Excepciones VI. Creación de excepciones propias. Vídeo 147. You Tube. (2015). Recuperado 20 Octubre 2020, de https://www.youtube.com/watch?v=pog50T99T6o&ab_channel=pildorasinformaticas.
- Garcia, J. (2015). Ejemplo de uso de Logger en Java. You Tube. Recuperado 20 Octubre 2020, de https://www.youtube.com/watch?v=86z50tHiqyc&ab_channel=JavierGarc%C3%ADaEscobedo.
- Framework vs Librería. You Tube. (2020). Recuperado 20 Octubre 2020, de <https://www.youtube.com/watch?v=A-iKX8Shge4>.
- Java log4j2 crear log diario. You Tube. (2017). Recuperado 20 Octubre 2020, de https://www.youtube.com/watch?v=zplpBTgZOYU&ab_channel=JavaFaces.
- Log4j – Download Apache Log4j 2. Logging.apache.org. (2020). Recuperado 20 Octubre 2020, de <https://logging.apache.org/log4j/2.x/download.html>.
- Depuración y gestión de logs. Jtech.ua.es. (2014). Recuperado 20 Octubre 2020, de <http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion06-apuntes.html>.

Anexos

Git Hub:

https://github.com/Apollo13-0/TareaExtra_1_Datos1.git