Semana 10

Excepciones en Java

Manejo de Excepciones

El manejo de excepciones en Java es uno de los medios efectivos para manejar los errores de tiempo de ejecución para que se pueda preservar el flujo regular de la aplicación. El manejo de excepciones de Java es un mecanismo para manejar errores de tiempo de ejecución como ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Definición de Excepción

La excepción es un evento no deseado o inesperado que ocurre durante la ejecución de un programa, es decir, en tiempo de ejecución, que interrumpe el flujo normal de las instrucciones del programa. Las excepciones pueden ser capturadas y manejadas por el programa. Cuando ocurre una excepción dentro de un método, crea un objeto. Este objeto se denomina objeto de excepción. Contiene información sobre la excepción, como el nombre y la descripción de la excepción y el estado del programa cuando se produjo la excepción.

Causas de Excepción

- Entrada de usuario no válida (la más común)
- Fallo del dispositivo
- Pérdida de conexión de red
- Limitaciones físicas (sin memoria de disco)
- Errores de código
- Abrir un archivo no disponible

Definición de Error

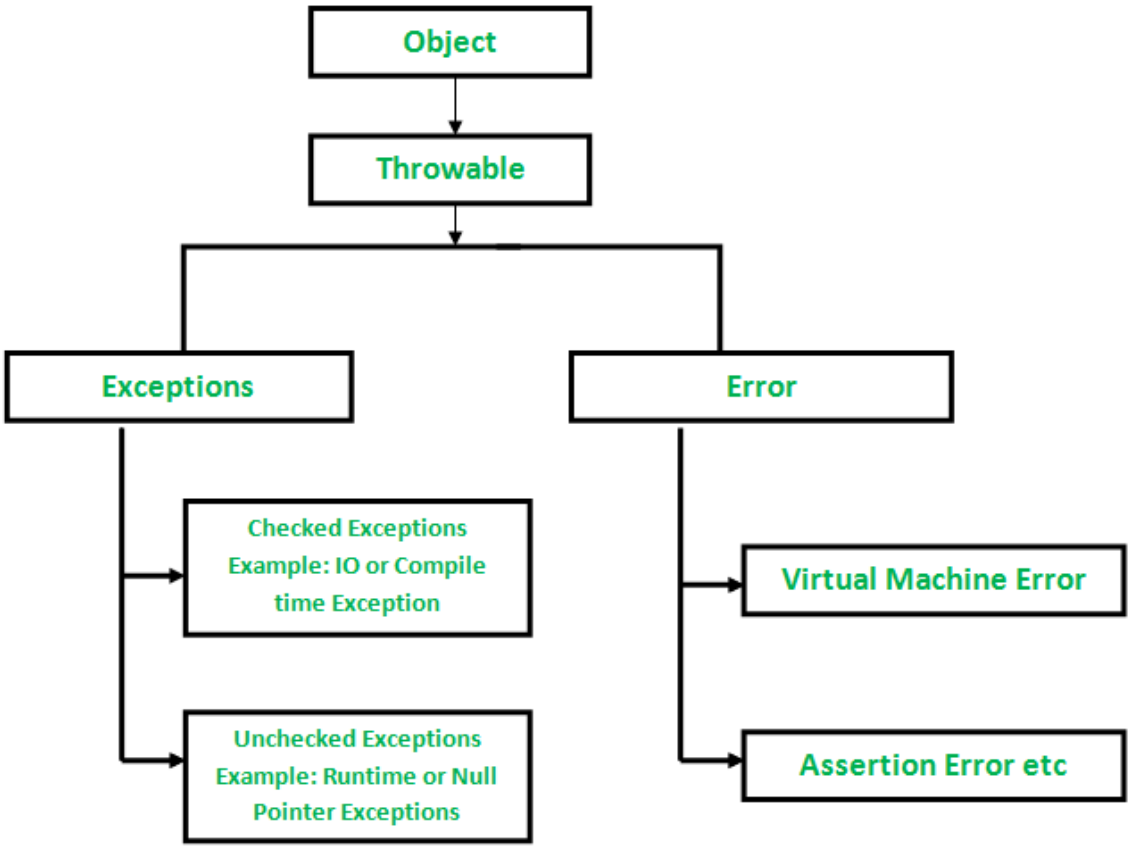
Los errores representan condiciones irrecuperables, como que la máquina virtual Java (JVM) se quede sin memoria, fugas de memoria, errores de desbordamiento de pila, incompatibilidad de bibliotecas, recursividad infinita, etc. Los errores suelen estar fuera del control del programador y no debemos tratar de manejarlos.

Diferencias entre Error y Excepción

- **Error:** un error indica un problema grave que una aplicación razonable no debería tratar de detectar.
- **Excepción:** la excepción indica condiciones que una aplicación razonable podría intentar capturar.

Jerarquía de Excepciones

Todos los tipos de excepciones y errores son subclases de la clase **Throwable**, que es la clase base de la jerarquía. Una rama está encabezada por **Exception**. Esta clase se utiliza para condiciones excepcionales que los programas de usuario deben detectar. NullPointerException es un ejemplo de tal excepción. Otra rama, **Error**, es utilizada por el sistema de tiempo de ejecución de Java (JVM) para indicar errores que tienen que ver con el propio entorno de tiempo de ejecución (JRE). StackOverflowError es un ejemplo de este tipo de error.



Sintaxis

1. **try:** El bloque try contiene un conjunto de declaraciones donde puede ocurrir una excepción.

```
try
{
    // declaraciones que pueden causar una excepción
}
```

2. **catch:** El bloque catch se usa para manejar la condición incierta de un bloque try. Un bloque de prueba siempre va seguido de un bloque de captura, que maneja la excepción que se produce en el bloque de prueba asociado.

```
catch
{
   // declaraciones que manejan una excepción
    // ejemplos, cerrar una conexión, cerrar
    // archivo, saliendo del proceso después de escribir
    // detalles a un archivo de registro.
}
```

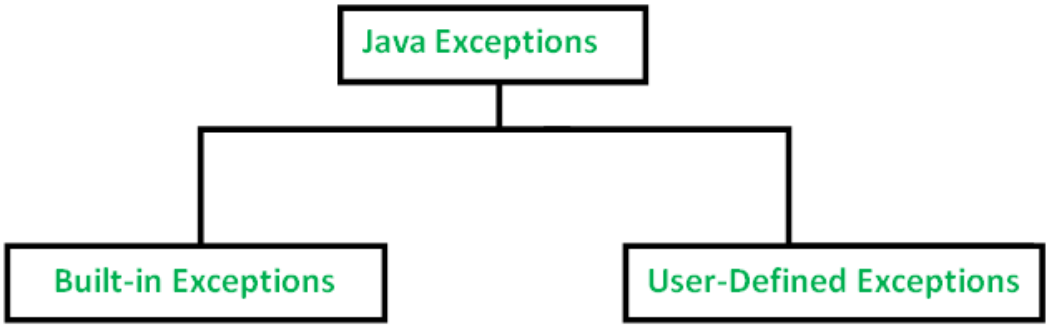
3. **throw:** La palabra clave throw se utiliza para transferir el control del bloque try al bloque catch.

4. **throws:** La palabra reservada throws se usa para el manejo de excepciones sin el bloque try & catch. Especifica las excepciones que un método puede lanzar a la persona que llama y no se maneja a sí mismo.

5. **finally:** Se ejecuta después del bloque catch. Lo usamos para poner código común (que se ejecutará independientemente de si se ha producido una excepción o no) cuando hay varios bloques catch.

Tipos de Excepciones

Java define varios tipos de excepciones que se relacionan con sus diversas bibliotecas de clases. Java también permite a los usuarios definir sus propias excepciones.



Excepciones Integradas en Java

Las excepciones integradas son las excepciones que están disponibles en las bibliotecas de Java. Estas excepciones son adecuadas para explicar ciertas situaciones de error. A continuación se muestra la lista de importantes excepciones integradas en Java.

1. **ArithmeticException:** Se lanza cuando se ha producido una condición excepcional en una operación aritmética.

2. **ArrayIndexOutOfBoundsException:** se lanza para indicar que se ha accedido a una matriz con un índice ilegal. El índice es negativo o mayor o igual que el tamaño de la matriz.

3. **ClassNotFoundException:** esta excepción se genera cuando intentamos acceder a una clase cuya definición no se encuentra.

4. **FileNotFoundException:** esta excepción se genera cuando no se puede acceder a un archivo o no se abre.

5. **IOException:** se lanza cuando una operación de entrada-salida falla o se interrumpe.

6. **InterruptedException:** se lanza cuando un subproceso está esperando, durmiendo o realizando algún procesamiento, y se interrumpe.

7. **NoSuchFieldException:** se lanza cuando una clase no contiene el campo (o variable) especificado.

8. **NoSuchMethodException:** Se lanza al acceder a un método que no se encuentra.

9. **NullPointerException:** esta excepción se genera cuando se hace referencia a los miembros de un objeto nulo. nulo no representa nada.

10. **NumberFormatException:** esta excepción se genera cuando un método no puede convertir una cadena en un formato numérico.

11. **RuntimeException:** Esto representa una excepción que ocurre durante el tiempo de ejecución.

12. **StringIndexOutOfBoundsException:** los métodos de la clase String la lanzan para indicar que un índice es negativo o mayor que el tamaño de la cadena.

13. **IllegalArgumentException:** esta excepción arrojará el error o la declaración de error cuando el método recibe un argumento que no se ajusta con precisión a la relación o condición dada. Viene bajo la excepción no verificada.

14. **IllegalStateException:** esta excepción generará un error o un mensaje de error cuando no se acceda al método para la operación particular en la aplicación. Viene bajo la excepción no verificada.

Ejemplo

```
// Programa Java para demostrar el funcionamiento de try,
// catch y finally

class Division {
    public static void main(String[] args)
    {
        int a = 10, b = 5, c = 5, result;
        try {
            result = a / (b - c);
            System.out.println("Resultado" + result);
        }

        catch (ArithmeticException e) {
            System.out.println("Excepcion capturada, division entre
cero");
        }

        finally {
            System.out.println("Impresion del bloque finally...");
        }
```

```
    }
}
```

Excepciones Definidas por el Usuario

A veces, las excepciones integradas en Java no pueden describir una situación determinada. En tales casos, el usuario también puede crear excepciones que se denominan "Excepciones definidas por el usuario".

Se siguen los siguientes pasos para la creación de una excepción definida por el usuario.

- El usuario debe crear una clase de excepción como una subclase de la clase de excepción. Dado que todas las excepciones son subclases de la clase Exception, el usuario también debe hacer de su clase una subclase de ella. Esto se hace como:

```
class MyException extends Exception
```

- Podemos escribir un constructor predeterminado en su propia clase de excepción.

```
MyException(){}
```

- También podemos crear un constructor parametrizado con una cadena como parámetro. Podemos usar esto para almacenar detalles de excepciones. Podemos llamar al constructor de superclase (Excepción) desde aquí y enviar la cadena allí.

```
MyException(String str)
{
   super(str);
}
```

- Para generar una excepción de un tipo definido por el usuario, necesitamos crear un objeto para su clase de excepción y lanzarlo usando la cláusula throw, como:

```
MyException me = new MyException(“Exception details”);
throw me;
```

Ejemplo

```
// Programa Java para demostrar el funcionamiento de throws.
```

```java
class ThrowsExecp {

    // Este método lanza una excepción
     // para ser manejada
     // por llamada o quien le llama
     // del que llama y así sucesivamente.
    static void fun() throws IllegalAccessException
    {
        System.out.println("Dentro de fun(). ");
        throw new IllegalAccessException("demo");
    }

    // Aquí está la llamada
    public static void main(String args[])
    {
        try {
            fun();
        }
        catch (IllegalAccessException e) {
            System.out.println("Capturado en main.");
        }
    }
}
```