# Excepciones

# Excepciones

Una excepción es una situación anómala que se da en la ejecución de un programa y que se supone que ocurre con poca frecuencia.

Para que un lenguaje trate excepciones debe proveer:

- ✔ Un modo de definirlas
- ✔ Una forma de alcanzarlas, invocarlas
- ✔ Una forma de manejarlas
- ✔ Un criterio de continuación

# Modelo de Manejo de excepciones

1. Java - Terminación

2. Python - Terminación

3. PL/1 - Reasunción

# Excepciones - Java

Modelo: por terminación. La unidad que generó la excepción busca manejar la excepción y termina.

Las excepciones que maneja el lenguaje son clases, las cuales extienden la clase Exception. Se instancian como cualquier clase, mediante el constructor new.

Existen una serie de excepciones ya definidas por el lenguaje (built-in exceptions, por ejemplo ClassNotFoundException, IOException, etc), y se permite al usuario crear nuevas excepciones (User defined exceptions)

Posee propagación dinámica (stack trace): una vez lanzadas, en caso de no ser tratadas por la unidad que las generó, se propagan dinámicamente. Si un método puede generar excepciones pero decide no manejarlas localmente, debe enunciarlas en su encabezado (throws)

Los bloques de código que manejan excepciones se distinguen por las palabras clave try y catch

Las excepciones se pueden lanzar explícitamente (throw), o se alcanzan por una condición de error (en el caso de las provistas por el lenguaje)

# Excepciones Java - Ejemplo

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
class File_notFound_Demo {
    private FileReader readFile() throws FileNotFoundException{
            // Following file does not exist
            File file = new File("E://file.txt");
            FileReader fr = new FileReader(file);
    }
    public static void main(String args[])  {
        try {
              this.readFile();
        }
```

```
catch (FileNotFoundException e) {
            System.out.println("File does
not exist");
             MyException me = new
MyException("Es una User-defined
exception");
             throw me;
         }
finally {
       System.out.println("Acá termina
el bloque try/catch");
    }
    }
}
```

# Excepciones - Java

Consideraciones:

- Puede haber tantos catch como excepciones maneje el método
- Se pueden anidar los bloques try/catch
- Si se omite el bloque catch, pero hay un bloque finally, este último se ejecuta antes de propagar la excepción en forma dinámica

# Excepciones - Python

- Maneja el modelo por terminación. El bloque que generó la excepción termina y busca manejarla
- Nuevamente el lenguaje provee una serie de excepciones predefinidas, y permite al programador definir las propias. El mecanismo es mediante la creación de clases que derivan de Exception
- Los bloques que manejan excepciones se identifican con try, y los manejadores con except.
- Posee dos tipos de propagación: en primer lugar, estática. Cuando la excepción ocurre dentro del bloque try, busca inmediatamente el manejador en los except siguientes. Si encuentra el manejador correspondiente, lo ejecuta y prosigue su ejecución en el bloque de código siguiente al try. En caso de no encontrar el manejador, lo propaga dinámicamente. Si llega al programa principal sin encontrar el manejador, termina en error.
- Las excepciones se pueden lanzar explícitamente con la palabra raise. Se usa raise nombre_excepcion. Si sólo aparece la palabra clave raise, se levanta anónimamente, se vuelve a lanzar la última excepción que estaba activa en el entorno actual. Si no hay ninguna, se levanta RuntimeError

# Excepciones - Python

Consideraciones:

- Puede haber tantos except como excepciones se manejen en el código (en un mismo except se pueden manejar varias excepciones)
- Puede haber un bloque else, que se ejecuta si no se levantó la excepción. También hay un bloque finally que se ejecuta siempre
- Así como en Java las excepciones posee su método stackTrace, en Python existe sys.exc_info()
- Los bloques try except se pueden anidar

# Excepciones - Python - Ejemplo

```
diccionario = {0:"Entrada1", 2:"Entrada2",
3:"Entrada3"}
try:
        print ('Entramos al bloque try')
        for x in range(1,6):
                if x == 2 or x == 3:
                        raise KeyError
                if x == 5:
                        raise CustomError
                else:
                        print (diccionario[x])
        print('Continuamos con el proceso..')
except KeyError as exc:
        diccionario[x] = 'NUEVO'
        #datos_exc = exc
        import sys
        print(sys.exc_info())
except:
        print("Manejo cualquier otra...")

        raise
else:
        print("Este mensaje se imprime porque NO se
levantó ninguna excepción")

finally:
        print("Este mensaje se imprime SIEMPRE")
diccionario
```

# Excepciones - PL/1

Modelo de REASUNCIÓN

Ningún proceso termina cuando se levanta una excepción, simplemente se detecta la excepción, se la atiende y continúa el flujo de ejecución en el punto siguiente a donde se levantó la excepción.

Se definen con la instrucción ON CONDITION nombre Manejador

Se levantan con la instrucción SIGNAL CONDITION nombre

Alcance de una excepción y búsqueda del manejador: a medida que se va ejecutando el proceso y se encuentran ON CONDITION se apila el manejador en una pila de manejadores, cuando se levanta una excepción se busca el nombre de la misma desde el tope de la pila hacia abajo, el primer manejador que se encuentra es el que se ejecuta.

Dentro de un bloque puede haber n cantidad de manejadores en cualquier lugar.

Excepciones con el mismo nombre se enmascaran entre sí.

# Excepciones - PL/1 - Ejemplo

```
Prog Main
        PROC UNO
        Begin
        ...
        ON CONDITION PEPE begin ... end;     ***DECLARACION y MANEJADOR3
        ..
        If (condError) then
                SIGNAL CONDITION PEPE     *** INVOCACION
        end
        ....
Begin
...
ON CONDITION PIPO begin ... end;         ***DECLARACION y MANEJADOR1
..
ON CONDITION PEPE begin ... end;         ***DECLARACION y MANEJADOR2
...
UNO;
...
If (condError) then
                SIGNAL CONDITION PIPO
end
```