# Excepciones

PROGRAMACIÓN

# Contenidos:

1) Excepciones.
2) Generación.
3) Captura.

# Excepciones

Las excepciones son el mecanismo por el cual pueden controlarse en un programa Java las condiciones de error que se producen. Estas condiciones de error pueden ser errores en la lógica del programa como un índice de un array fuera de su rango, una división por cero o errores disparados por los propios objetos que denuncian algún tipo de estado no previsto, o condición que no pueden manejar.

La idea general es que cuando un objeto encuentra una condición que no sabe manejar crea y dispara una excepción que deberá ser capturada por el que le llamó o por alguien más arriba en la pila de llamadas. Las excepciones son objetos que contienen información del error que se ha producido y que heredan de la clase Throwable o de la clase Exception. Si nadie captura la excepción interviene un manejador por defecto que normalmente imprime información que ayuda a encontrar quién produjo la excepción.

Existen dos categorías de excepciones:

- **Excepciones verificadas**: El compilador obliga a verificarlas. Son todas las que son lanzadas explícitamente por objetos de usuario.
- **Excepciones no verificadas**: El compilador no obliga a su verificación. Son excepciones como divisiones por cero, excepciones de puntero nulo, o índices fuera de rango.

# Generación de excepciones

```
public class Empresa
{
   String nombre;
   Empleado [] listaEmpleados;
   int totalEmpleados = 0;
   . . .
   Empresa(String n, int maxEmp)
  {
      nombre = n;
      listaEmpleados = new Empleado [maxEmp];
   }
   . . .
   void nuevoEmpleado(String nombre, int sueldo)
   {
     if (totalEmpleados < listaEmpleados.length )
     {
         listaEmpleados[totalEmpleados++] = new Empleado(nombre,sueldo);
     }
   }}
```

El método nuevoEmpleado comprueba si hay sitio en el array para almacenar la referencia al nuevo empleado. Si lo hay se crea el objeto. Pero si no lo hay el método no hace nada más. No da ninguna indicación de si la operación ha tenido éxito o no.

Opciones:

- Se podría hacer una modificación para que, por ejemplo el método devolviera un valor booleano true si la operación se ha completado con éxito y false si ha habido algún problema.
- Otra posibilidad es generar una excepción verificada (Una excepción no verificada se produciría si no se comprobara si el nuevo empleado va a caber o no en el array).

# Generación de excepciones

```
void nuevoEmpleado(String nombre, int sueldo) throws
CapacidadEmpresaExcedida
{
     if (totalEmpleados < listaEmpleados.length)
    {
        listaEmpleados[totalEmpleados++] = new
Empleado(nombre,sueldo);
     }
     else throw new CapacidadEmpresaExcedida(nombre);
 }
```

```
public class CapacidadEmpresaExcedida extends Exception
{
   CapacidadEmpresaExcedida(String nombre)
{
      super("No es posible añadir el empleado " + nombre);
   }
}
```

1) Las excepciones son clases, que heredan de la clase genérica **Exception**.
2) Es necesario por tanto asignar un nombre a nuestra excepción. Se suelen asignar nombres que den alguna idea del tipo de error que controlan.
3) Para que un método lance una excepción:
   - Debe declarar el tipo de excepción que lanza con la cláusula **throws**, en su declaración.
   - Debe lanzar la excepción, en el punto del código adecuado con la sentencia **throw**.

# Captura de excepciones

**Sin excepciones**, invocaríamos el método nuevoEmpleado de la siguiente forma:

```
Empresa em = new Empresa("La Mundial");
em.nuevoEmpleado("Adán Primero",500);
```

**Con excepciones**, tenemos que crear un bloque try/catch y, por lo tanto, recibiríamos información sobre lo ocurrido:

```
Empresa em = new Empresa("La Mundial");
try
{
   em.nuevoEmpleado("Adán Primero",500);
} catch (CapacidadEmpresaExcedida exc)
{
   System.out.println(exc.toString());
   System.exit(1);
}
```

**Nota**: Sino añades el bloque try/catch, la excepción no sería captura y el compilador lanzaría un error de compilación.

# Captura de excepciones

En resumen, el formato general del bloque try / catch es:

```
try
{
   . . .
} catch (Clase_Excepcion nombre) { . . .}
  catch (Clase_Excepcion nombre) { . . .}
//Esto permite capturar varios tipos de Excepciones.
```

En ocasiones el código que llama a un método que dispara una excepción tampoco puede (o sabe) manejar esa excepción. Si no sabe que hacer con ella… ¿la volvemos a lanzar?
Una opción alternativa, no la más apropiada, es que nuestro main lance la excepción.

```
public static void main(String [] args) throws CapacidadEmpresaExcedida
{
   Empresa em = new Empresa("La Mundial");
   em.nuevoEmpleado("Adán Primero",500);
}
```

# Finally

En resumen, el formato general del bloque try / catch es:

```
try
{
   . . .
} catch (Clase_Excepcion nombre) { . . .}
  catch (Clase_Excepcion nombre) { . . .}
//Esto permite capturar varios tipos de Excepciones.
  finally{….}
```

Por último, podemos añadir la cláusula finally tras los catchs. Esta sirve para especificar un bloque de código que se ejecutará tanto si se lanza una excepción como si no. Suele usarse para limpiar el estado interno de los objetos afectados o para liberar recursos externos (descriptores de fichero, por ejemplo