

Excepciones

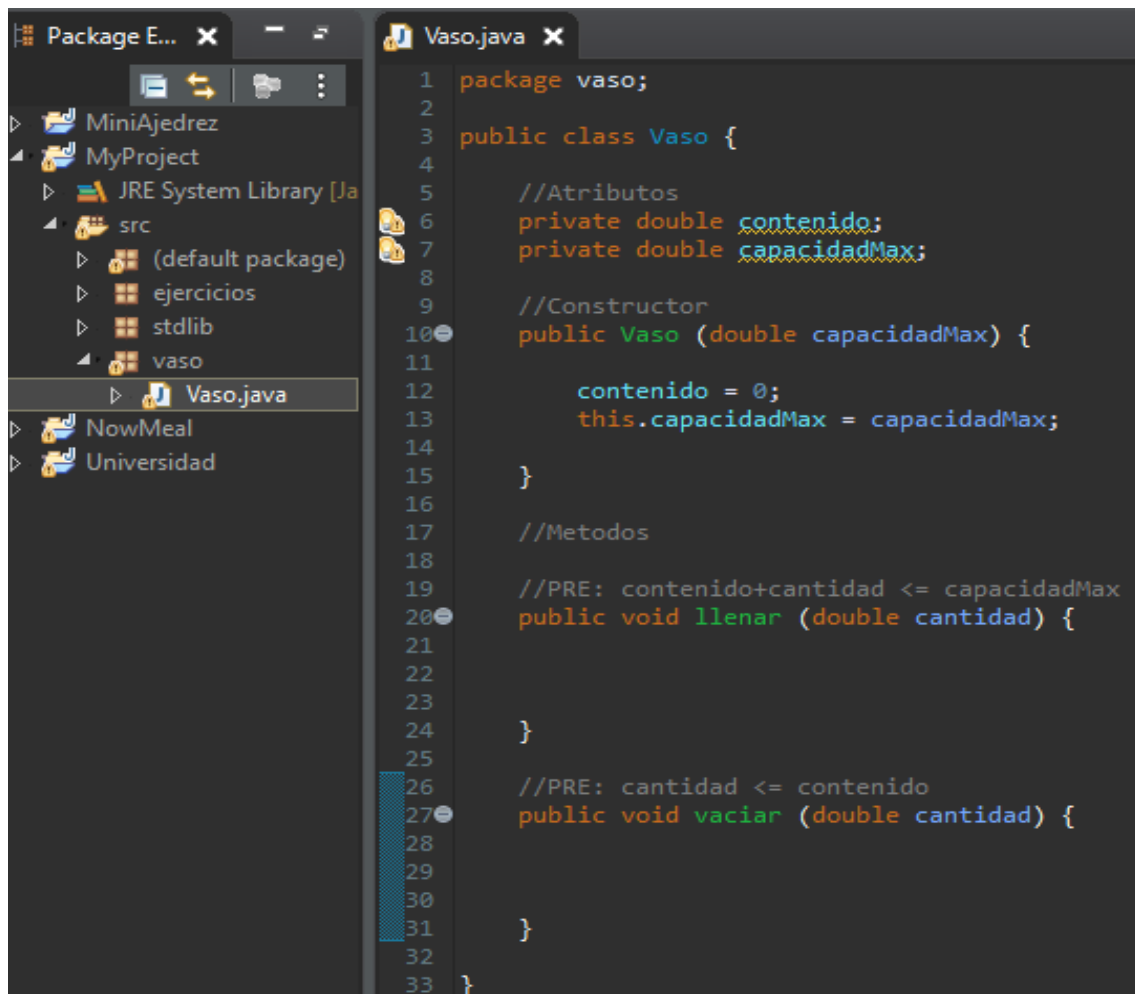
Las excepciones sirven para notificar una situación anómala, y ocurren **siempre** durante la ejecución del programa:

```
int[] nums = {0,1,2};  
System.out.println(nums[3]);
```

>>>

```
Console x  
terminated> Pruebas [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (1 ju  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3  
at Pruebas.main(Pruebas.java:8)
```

Existen muchas excepciones que vienen predefinidas por Java, pero si, por ejemplo, queremos crear una clase orientada a objetos que represente un **Vaso**, con la siguiente forma:

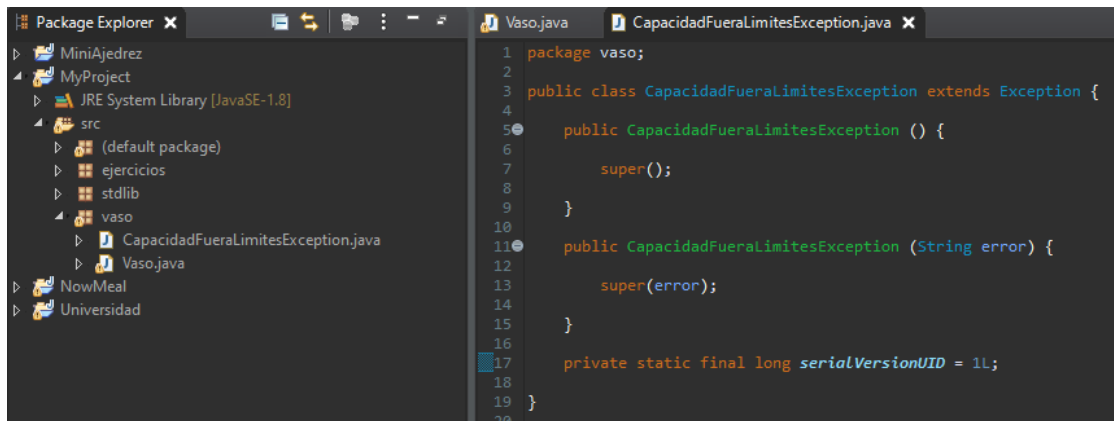


```
1 package vaso;  
2  
3 public class Vaso {  
4  
5     //Atributos  
6     private double contenido;  
7     private double capacidadMax;  
8  
9     //Constructor  
10    public Vaso (double capacidadMax) {  
11  
12        contenido = 0;  
13        this.capacidadMax = capacidadMax;  
14    }  
15  
16    //Metodos  
17  
18    //PRE: contenido+cantidad <= capacidadMax  
19    public void llenar (double cantidad) {  
20  
21  
22  
23    }  
24  
25    //PRE: cantidad <= contenido  
26    public void vaciar (double cantidad) {  
27  
28  
29  
30  
31    }  
32  
33 }
```

Ahora bien, los metodos llenar() y vaciar() tienes una PREcondición cada uno, **vamos a crear una excepción** llamada CapacidadFueraLimitesException.

Como crear excepciones

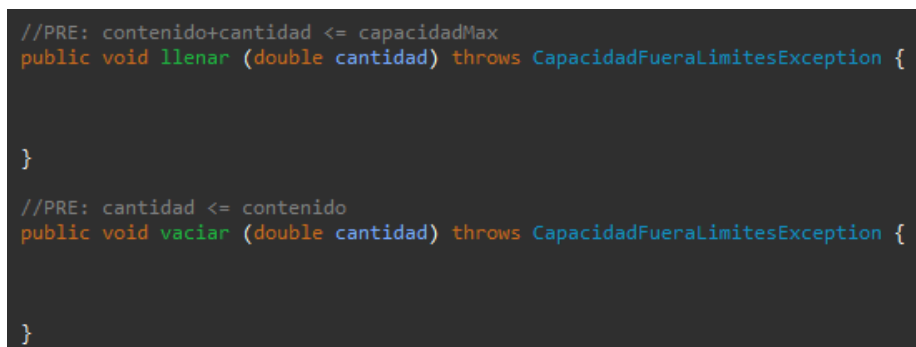
Las excepciones en java son **Objetos**, y son una subclase de Exception, en nuestro caso, CapacidadFueraLimitesException tendría ésta forma:



```
1 package vaso;
2
3 public class CapacidadFueraLimitesException extends Exception {
4
5     public CapacidadFueraLimitesException () {
6
7         super();
8     }
9
10    public CapacidadFueraLimitesException (String error) {
11
12        super(error);
13    }
14
15    private static final long serialVersionUID = 1L;
16
17 }
18
19
20
```

Como veis, creo un .java en el paquete que se encuentra Vaso.java, llamado CapacidadFueraLimitesException, es indiferente que se encuentren el mismo paquete o no, pero hay que tener en cuenta que para poder utilizarlo debemos respetar la visibilidad de nuestra clase, de tal forma que si es necesario, tendremos que añadirla a los imports de Vaso.java.

Ahora que tenemos creada la excepción, iremos donde esos dos métodos que pueden dar problemas y se lo asignaremos de ésta forma:



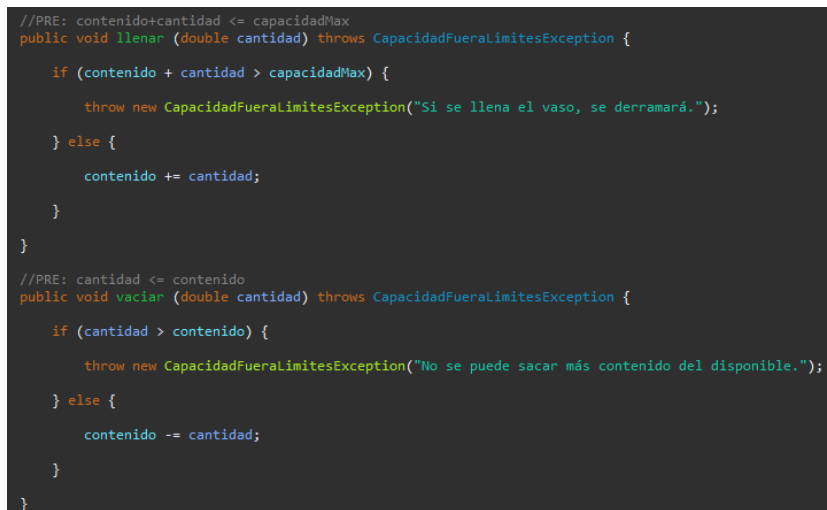
```
//PRE: contenido+cantidad <= capacidadMax
public void llenar (double cantidad) throws CapacidadFueraLimitesException {

}

//PRE: cantidad <= contenido
public void vaciar (double cantidad) throws CapacidadFueraLimitesException {

}
```

Y por último, hacemos que lance la excepción si se dan las condiciones:



```
//PRE: contenido+cantidad <= capacidadMax
public void llenar (double cantidad) throws CapacidadFueraLimitesException {

    if (contenido + cantidad > capacidadMax) {

        throw new CapacidadFueraLimitesException("Si se llena el vaso, se derramará.");

    } else {

        contenido += cantidad;

    }

}

//PRE: cantidad <= contenido
public void vaciar (double cantidad) throws CapacidadFueraLimitesException {

    if (cantidad > contenido) {

        throw new CapacidadFueraLimitesException("No se puede sacar más contenido del disponible.");

    } else {

        contenido -= cantidad;

    }

}
```

Tratamiento de excepciones: TRY – CATCH - FINALLY

Cuando realizamos lo anterior, no estamos tratando la excepción, solo estamos diciendo que no se puede realizar la tarea, pero ahora imaginemos que queremos hacer lo siguiente:

- Si en llenar, ponemos una cifra que sobrepase la capacidad máxima, en vez de lanzar la excepción, llenamos el vaso hasta arriba.
- Si en vaciar, ponemos una cifra mayor que el contenido actual del vaso, en vez de lanzar la excepción, vaciamos el vaso.

Esto se puede realizar con un try catch finally, y nos quedaría así:

```
//PRE: contenido+cantidad <= capacidadMax
public void llenar (double cantidad) {
    try {
        comprobacion(cantidad);
    } catch (CapacidadFueraLimitesException c) {
        cantidad = capacidadMax - contenido;
    } finally {
        contenido += cantidad;
    }
}

private void comprobacion (double cantidad) throws CapacidadFueraLimitesException {
    if (contenido + cantidad > capacidadMax) throw new CapacidadFueraLimitesException("Si se llena el vaso, se derramará.");
}
```

En primer lugar he creado una función auxiliar llamada comprobacion, que nos determina si la cantidad añadida excede la capacidad máxima, y si es así, lanza la excepción.

Hago el **try** de comprobacion, sino suelta la excepción, va directamente al **finally**, pero si la suelta, nos metemos en **catch**, donde ajusto la cantidad, para que sea exactamente lo que falta para llegar al máximo, una vez pasamos por el **catch**, vamos al **finally**.

La sentencia **finally**, se ejecuta **SIEMPRE**, es indiferente que salte la excepción o no, siempre se va a ejecutar.

Aquí he hecho una exageración, para que se vea como funciona, el caso es que por ejemplo, al utilizar un **Stack** (PDF 16), podemos usar dos funciones, **peek()** y **pop()**, y ambos pueden soltar la excepción `EmptyStackException`.

Por lo que si usamos peek o pop, tenemos que hacer un try catch de esa excepción, obligatoriamente, como se ve aquí:

```
try {
    //le asigno al resultado el primer movimiento de la pila de deshacer
    resultado = pilaDeshacer.peek();

    //borro el movimiento de deshacer y lo añado a rehacer
    pilaRehacer.push(pilaDeshacer.pop());
} catch (EmptyStackException e) {
    Logger.getLogger("Test").log(Level.INFO, "BOOM!", e);
}
```

Como veis, al utilizar las funciones `peek()` y `pop()` tengo que realizar un `try catch` de `EmptyStackException` (no necesitáis saber que he hecho en el `catch`).

También podemos no usar la sentencia **finally**, el caso es que si, toda nuestra función se va a basar en que hacer si salta una excepción o no, es recomendable usar dicha sentencia.

Podemos realizar varios `catch` en un solo `try`, y poner dos en la misma línea.

```
//PRE: contenido+cantidad <= capacidadMax
public void llenar (double cantidad) {

    try {

        comprobacion(cantidad);

    } catch (CapacidadFueraLimitesException c) {

        cantidad = capacidadMax - contenido;

    } catch (ArrayIndexOutOfBoundsException a) {

        /*
         * codigo
         */

    } finally {

        contenido += cantidad;

    }

}

private void comprobacion (double cantidad) throws CapacidadFueraLimitesException {

    if (contenido + cantidad > capacidadMax) throw new CapacidadFueraLimitesException("Si se llena el vaso, se derramará.");

}
```

Aquí pongo dos `catch`.

```
try {

    comprobacion(cantidad);

} catch (CapacidadFueraLimitesException | ArrayIndexOutOfBoundsException c) {

    cantidad = capacidadMax - contenido;

} finally {

    contenido += cantidad;

}
```

Aquí pongo dos excepciones en la misma línea, separados por una `|`.

```
private void comprobacion (double cantidad) throws CapacidadFueraLimitesException, ArrayIndexOutOfBoundsException{

    if (contenido + cantidad > capacidadMax) throw new CapacidadFueraLimitesException("Si se llena el vaso, se derramará.");

}
```

Aquí los pongo a ambos, para que los pueda throwear un solo método.

No tiene sentido esto último que he hecho, sobre realizar varios **catch** en un solo **try**, pero era para que se entendiese mejor, como podíamos llevar a cabo esto.