Excepciones

Lanzar y cachar

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

1 de diciembre de 2020

Definiciones

1. Definiciones
2. Implementación en Java
3. Lanzar y cachar

Temas

1. Definiciones
   - Excepción
   - Propagación

Excepción

Definición (Excepción)

"Una *excepción* es un evento que ocurre durante la ejecución de un programa que altera el flujo normal de instrucciones."
Gallardo y col. 2020

- En `Java` los métodos donde ocurren errores crean **objetos** de tipo `Exception` que pasan al sistema de ejecución (*runtime system*).
- Estos objetos contienen información sobre el error ocurrido, en particular:
  - El tipo de error.
  - El estado del programa cuando ocurrió el error.

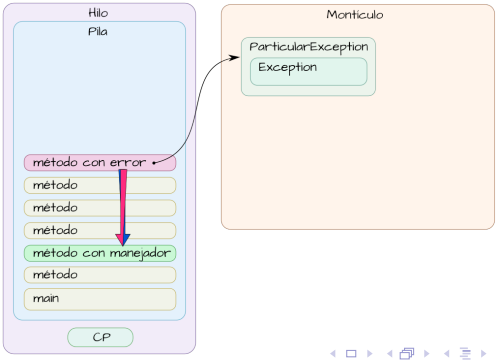
Temas

1. Definiciones
   - Excepción
   - Propagación

Propagación de la excepción

- Cuando un método *lanza* una excepción, indicando que ha ocurrido un error, esta se propaga por la pila de ejecución desmontando registros hasta ser atendida por un método que sepa qué hacer.

- La secuencia de métodos desmontados se almacena en la *traza de la pila* (*stacktrace*).



Implementación en Java

1. Definiciones
2. Implementación en Java
3. Lanzar y cachar

Temas

2 Implementación en Java
- Tipos de Errores
- Jerarquía

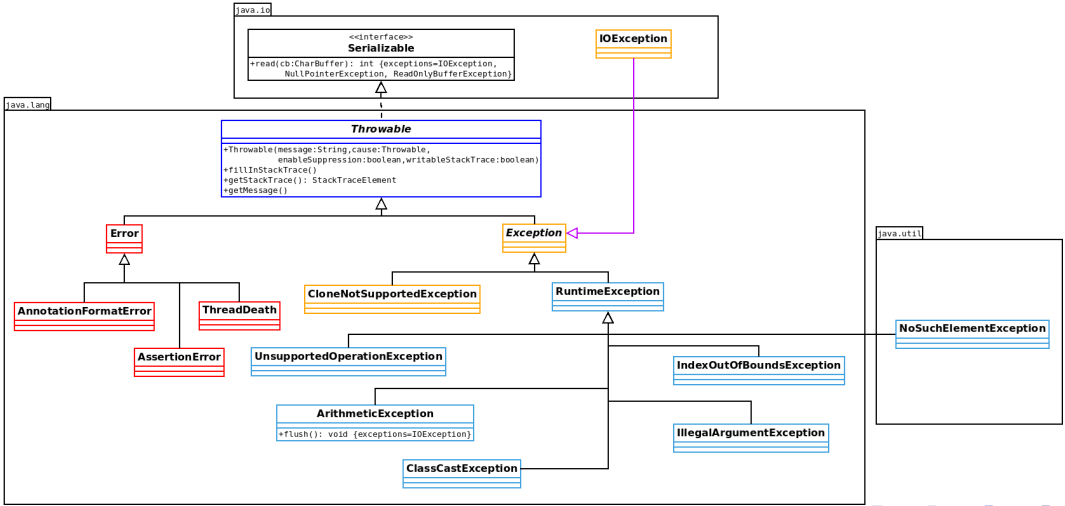
Tipos de Errores

- *Excepciones.* Objetos de tipo `Exception` indican errores de menor gravedad y usualmente son cachados y antendidos por alguna función dentro de la traza de la pila.
  Se distinguen dos tipos:
  - *Excepciones controladas.* Son aquellos errores de los cuales una buena aplicación se debe poder recuperar.
    Cualquier método que las lance debe notificar que esa excepción puede ocurrir.
  - *Excepciones no controladas.* Indican *bugs* en el programa como errores lógicos o usos inadecuados de una API (no cumplir con las precondiciones de una función).
    - Heredan de la clase `RuntimeException`.
    - Se pueden atrapar, pero es mejor programar de tal manera que no sean lanzadas.
- *Errores.* Objetos de tipo `Error` indican fallos en el funcionamiento de la máquina virtual y típicamente no son cachados por los programas.
  También se les considera *Excepciones no controladas*.

Temas

2 Implementación en Java
- Tipos de Errores
- Jerarquía

Jerarquía, algunos ejemplos



Lanzar y cachar

1 Definiciones

2 Implementación en Java

3 Lanzar y cachar

Temas

3 Lanzar y cachar
- Lanzar

Lanzar

- Se *lanza* una excepción cuando ha ocurrido un error en el programa.
- Para ello se usa el comando `throw`.
- Si la excepción es controlada, el método que la lanza debe avisarlo en su firma utilizando `throws`.

```
1 public void trabajaConFlujo() throws IOException {
2   // ...
3   throw new IOException("Algo␣salió␣mal␣al␣leer␣los␣datos");
4   // ...
5 }
```

Cachar

- Un método que manda llamar a otro, que puede lanzar excepciones, puede *atrapar* o *cachar* la excepción y atender el problema.
- Se pueden atrapar varias excepciones, pero es importante intentar atrapar primero a las más específicas y al final a las más generales, porque el sistema ejecuta el código del primer `catch` que caze con el tipo de la excepción.
- Si necesitamos ejecutar un código, pase lo que pase, se puede utilizar el bloque `finally`.

Ejemplo

Listing 1: Try, catch, finally

```
1  public void ejecutaOperación(Comando comando) {
2    try {
3        switch(comando) {
4            case suma:
5                System.out.println(op1.suma(op2));
6                break;
7            case resta:
8                System.out.println(op1.resta(op2));
9        }
10   } catch(UnsupportedOperationException uoe) {
11       System.out.println("No está programada, intenta con otra.");
12   } catch(ConversionNoSoportadaException cns) {
13       System.out.println("No es válido realizar esta operación.");
14   } catch(Exception e) {
15       System.out.println("Algo está mal.");
16   } finally {
17       System.out.println("Haya o no excepción, esto se ejecuta.");
18   }
19
20 }
```

Atrapar con recursos

- Es un atajo para utilizar *recursos*, objetos que deben ser liberados, haya error o no mediante una llamada al método `close`.
- Son objetos que implementan las interfaces `java.lang.AutoCloseable` y/o `java.io.Closeable`.
- Nos ahorra el bloque `finally` si lo único que queríamos hacer era cerrar el recurso.
- Pueden, además, tener `catch` y `finally` si se requiere y estos se ejecutan después de haber cerrado el recurso.

Ejemplo

Listing 2: Vieja escuela

```
1  public String lee(String ruta) throws IOException {
2      BufferedReader br = new BufferedReader(new FileReader(path));
3      try {
4          return br.readLine();
5      } catch (IOException) {
6          System.out.println("No se pudo leer.");
7      } finally {
8          if (br != null) br.close();
9      }
10 }
```

Listing 3: Con recursos

```
1  public String lee(String ruta) throws IOException {
2      try (BufferedReader br = new BufferedReader(new FileReader(path))) {
3          return br.readLine();
4      } catch (IOException) {
5          System.out.println("No se pudo leer.");
6      }
7  }
```

Bibliografía I


Gallardo, Raymond y col. (1 de dic. de 2020). *Lesson: Exceptions*. Ed. por Oracle.
URL: https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html.

Licencia

Creative Commons
Atribución-No Comercial-Compartir Igual