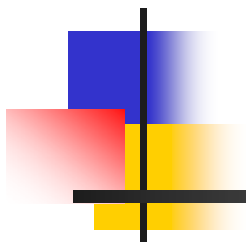
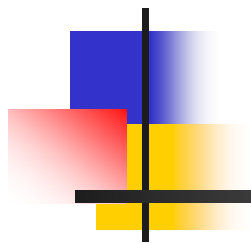


# Práctica 3





### 3. Excepciones



# Excepciones

- En la práctica 3 hay que crear seis nuevas clases de excepciones: **DivByZeroException**, **StackException**, **ExecutionErrorException**, **ArrayException**, **BadFormatByteCodeException** y **LexicalAnalysisException**
- Las dos primeras heredan de la tercera. Las cuatro últimas heredan de la clase predefinida **Exception**
- Además se usan dos clases de excepciones predefinidas: **NumberFormatException** y **FileNotFoundException**
- Recuerda que las clases de excepciones predefinidas NO hay que crearlas; el resto, sí.



# DivByZeroException

---

- Es la que se lanza cuando se va a dividir la cima entre la subcima de la pila de operandos, y la subcima es 0
- Se lanza en **operates()** de **Div** y en quien llame a este método y no capture la excepción (por ejemplo, **execute()** de **Arithmetics**, y obviamente **execute()** de la interfaz **Bytecode**)
- Hereda de **ExecutionErrorException** y es tratada cuando se tratan excepciones de esta superclase



# StackException

---

- Se lanza por dos motivos: cuando se quiere desapilar de la pila de operandos y está vacía (**empty stack exception**) y cuando se intenta apilar y está llena (**stack overflow exception**)
- La lanzan pues aquellos métodos que intenten sacar o añadir información de la pila de operandos. Por ejemplo, **push()** y **pop()** de **OperandStack**; **execute()** de **Push**, **Store**, **Load** y **Out**
- Hereda de **ExecutionErrorException** y es tratada cuando se tratan excepciones de esta superclase



# ExecutionErrorException

---

- Se lanza cuando, al ejecutar un bytecode, se produce una excepción (división por cero, desapilar de pila vacía, apilar en pila llena)
- Es una clase con dos subclases:  
**DivByZeroException** y **StackException**
- La lanzan, por ejemplo, **run()** de **CPU**, **execute()** de **Run**, **executeRun()** de **Engine**
- Es capturada en **start()** de **Engine**



# ArrayException

---

Se lanza cuando se intenta acceder a posiciones incorrectas de un array; por ejemplo, al añadir algo a un array que está lleno, o al acceder a una componente que no existe de un array

- La lanza **execute()** de **Command** y, por tanto, algunas de las clases que implementan esta interfaz; por ejemplo, **Load** (cuando añade strings al programa fuente), **Replace** (cuando reemplaza bytecodes del programa de bytecodes), **Run** (y, por tanto, **Engine** y **CPU**; cuando accede a instrucciones del programa de bytecodes) y **Compile** (cuando compila)
- La lanza **compile()** de **Instruction** y, por tanto, todas las instrucciones que implementan esta interfaz (**SimpleAssignment**, **IfThen**, **While**, **CompoundAssignment**, **Return**, **Write**) más alguna relacionada como **Condition**, cuando añaden bytecodes en la compilación



# ArrayException

---

- Otras clases que la lanzan son **Compiler** (cuando accede a instrucciones del programa parseado), **ByteCodeProgram**, **SourceProgram** y **ParsedProgram** (cuando acceden a sus componentes)
- **Engine** la lanza y captura en **load()** porque no cabe el archivo leído en el programa fuente
- Es capturada en **start()** de **Engine**





# LexicalAnalysisException

---

- Se lanza cuando se encuentran errores en el parseo de un programa fuente (falta la línea **end**, por ejemplo, o una línea es incorrecta)
- La lanza **execute()** de **Command** y, de las clases que implementan esta interfaz, la lanza **Compile**
- La lanza evidentemente **lexicalParser()** de la clase **LexicalParser**
- Hay clases que la lanzan y la capturan, como **IfThen** y **While**
- Es capturada en **start()** de **Engine**, informando de la línea que es incorrecta



# BadFormatByteCodeException

---

- Se lanza cuando se encuentra un bytecode incorrecto
- Observa que, en la práctica 3, los bytecodes son (casi siempre) generados por lo que solo la lanza **execute()** de **Replace**, que es la clase encargada de leer bytecodes
- Es capturada en **start()** de **Engine**



# Otras excepciones

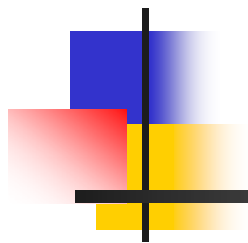
---

- **NumberFormatException**: excepción predefinida que se lanza cuando se esperaba un número y se lee algo diferente
  - No es necesario importarla
  - Se lanza y captura en **parse()** de **Number** y **Replace**
- **FileNotFoundException**: excepción predefinida que se lanza cuando no se encuentra el archivo cuyo nombre se ha suministrado
  - Se importa con el paquete **java.io**
  - La lanza **load()** de **Engine** y **execute()** de **Load**, y la captura **start()** de **Engine**

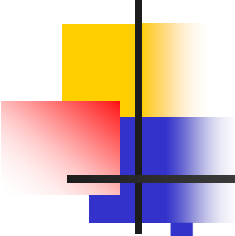


# Consideraciones generales

- Consulta el doc y observa que si un método tuviera que lanzar una excepción (porque contiene una llamada a otro método que la lanza) y no lo hiciera es porque esa excepción la captura con un bloque **try/catch**
- Siendo cierto lo anterior recuerda que, cuando un método sobrescribe otro, puede lanzar menos, pero no más, excepciones
- Una excepción se puede capturar (para informar de por qué se ha producido) y volver a lanzar, en el mismo bloque **catch** (para informar después, cuando se tenga otra información). Por ejemplo, se puede capturar una excepción por un intento de acceso a cima de pila vacía, y lanzar la misma excepción de nuevo para informar después, cuando se capture, de qué instrucción intentó el acceso
- Mediante tratamiento de excepciones debes conseguir que se puedan almacenar, compilar y ejecutar archivos, de forma consecutiva



## 4. Lectura de archivos

- 
- Se crea **arch.txt** respetando lo dicho en la sección 1 del enunciado de la práctica 3 (no hay líneas en blanco, ...)
  - Se guarda junto con las carpetas **src**, **doc**, ...
  - En **Engine**, se importan las clases:  
**java.io.File**  
**java.io.FileNotFoundException**
  - En **load()** de **Engine** se hace:  
**Scanner sc = new Scanner(new File(arch));**
  - Se usa este scanner para leer las líneas del archivo y almacenarlas en el **sProgram** de **Engine**
  - Al final se cierra **sc**, haya habido excepciones o no