Effective Java Exceptions

By Barry Ruzek

http://dev2dev.bea.com/pub/a/2006/11/effective-exceptions.html?page=1

Index

Why Exceptions Matter

- Java exceptions
  - methods communicate alternative outcomes for an operation
  - deserve special attention in your application architecture
  - catching, logging, trying to determine, translating one exception to another
- Exception handling code
  - represents skill of a java architect
  - how much code
  - clean, compact, coherent exception handling ->consistent approach to using java exception
  - Exception handling are very predictable

Why Exceptions Matter

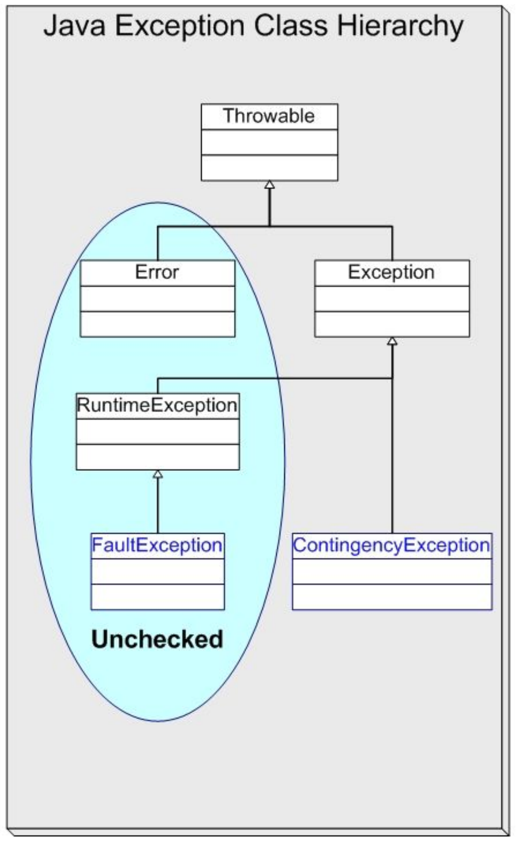
- Questionare
  - why do you add the exception code in the application ?
  - what would happen when exception actually occurred ?
- Java component error reporting design
  - flawing reporting : when and why
    - "log and forget" code
  - twisted logic path
    - nested try/catch/finally blocks

keeping application simple, maintainable, correct

Challenging the Exception Canon

- Java Exception Model
  - first language in which the compiler enforced exception rules
  - exception rules
    - Method :
      - exception based on java.lang.Throwable
      - "throws"
    - Client :
      - catch or throw
- “unchecked exception”

  vs “checked exception”
- compile time type exception checking
  - prevent nasty surprise at runtime
  - produce error-free software



Challenging the Exception Canon

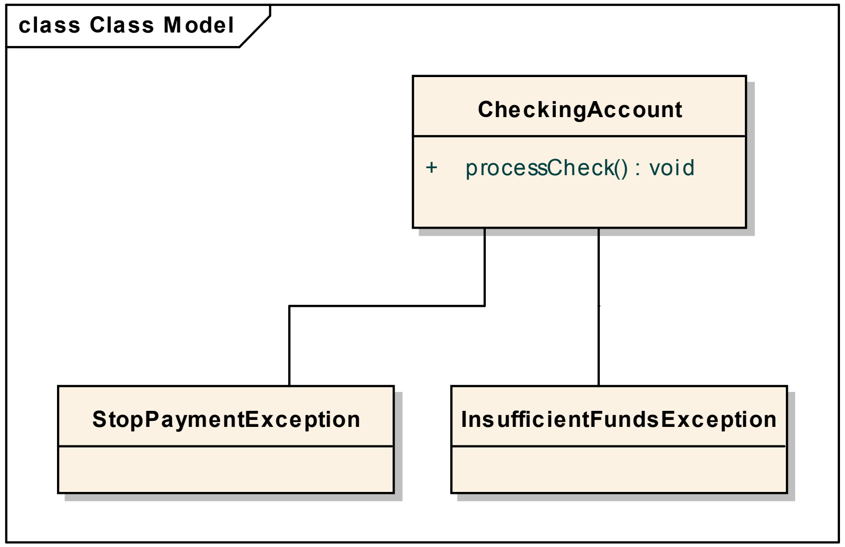
- declared checked exception for every possible failure
  - extensive Java Library API
  - Example :
    - java.io.IOException used in 63 java library package
  - Problem
    - I/O failure occurred rarely
    - there is usually nothing your code can do to recover from one
    - not catching them was worse since the compiler required
- No-win situation
  - anti patterns
  - question
    - whether java's checked exception model was a failed experiment.
    - failure was in the thinking by the JAVA API designers

Faults and Contingencies

- Example : Banking application

CheckingAccount

Owner : customer
Behavior :
  1.maintains a current balance
  2.accept deposit
  3.accept stop payment order on
    checks prcoess incoming checks
Contingencies :
  1.stop payment order registered for
the check
  2.account may no have sufficient
funds to cover the check amount



Faults and Contingencies

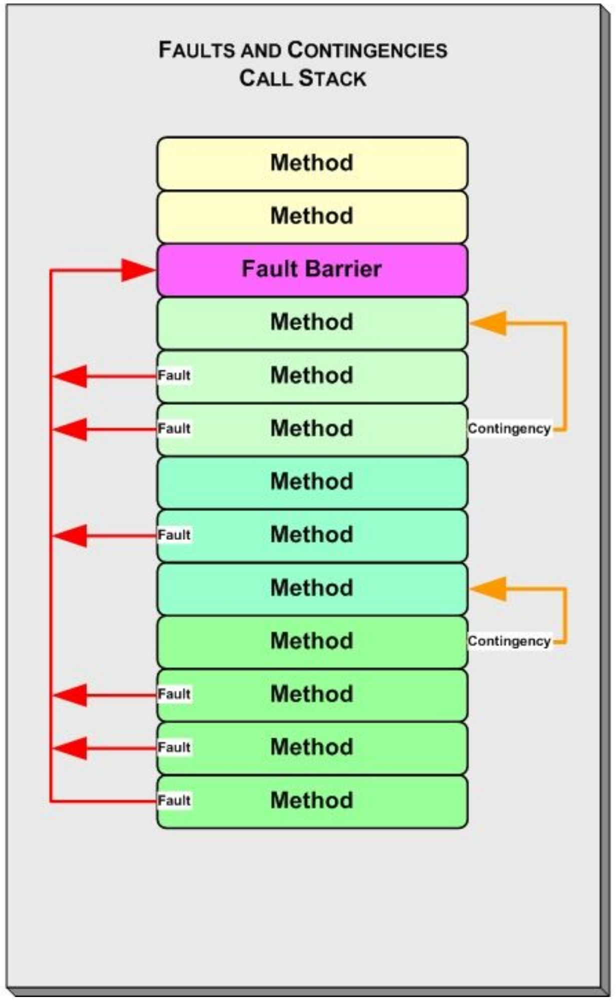
- Question
  - do not represent a failure of the software or of the execution environment
    - turn on the database server
    - unplugged a network
    - changed the password needed to access the database
  - JDBC
    - SQLException
    - method should know about its own implementation
- Contingency & Fault
  - Contingency : expected condition demanding and alternative response from a method
  - Fault : unplanned condition

Fault handling

- Use unchecked exception for fault condition
  - Java runtime mistake
    - RuntimeException
      - Example : ArithmeticExceptin, ClassCastException
  - Faulting handling strategy
    - checked exception in fault conditons
    - Multiple fault exception type is bad.
    - Single fault exception with a good descriptive message is good.
    - Java exception chaining (as of Java 1.4)
- Establish a fault barrier
  - Not interesting to software logic but to people
  - All methods on the call stack to a level designed to catch them
  - Fault-generating method is not required to declare and catch them

Fault handling

- Fault Handling
  - Minimize code clutter
  - Capture and preserve diagnostics
  - Alert the right person
  - Exit the activity gracefully
- Establish a fault barrier
  - Separation of concerns
  - Fault barrier pattern
    - Any component can throw a fault exception, but only "fault barrier" catchs them.
    - Fault barrier resides the top of the call stack



Fault handling

- Fault barrier role
  - record information in the fault exception
  - close out the operation in a controlled manner
    - Ex) SOAP <fault/>, generic HTML response
- Fault barrier class
  - Sruts framework
    - org.apache.struts.action.ExceptionHandler
  - Spring MVC framework
    - SimpleMappingExceptionResolver
      - resolveException()

Contingency Handling

- simple contingencies
  - using method's return type
    - null : null instead of actual object, null or not null
    - integer value like -1
    - alternative return value
  - void return type : exception
- complex contingencies
  - integral value
  - error code
- supply something useful
  - convey diverse conditions to method caller
  - complete java type
- to log or not to log
  - contingency represents a significant event
    - log entry recording is useful
  - every exception in rare event
    - no actual benefit

Exception Aspects

- common conventions
  - fault barrier method must reside at the head of a graph of methods
  - all use unchecked exception to signify fault conditions
  - specific unchecked exception type
    - fault barrier is expecting to receive
  - catch and translate checked exception from lower methods that are deemed to be faults in their execution context
  - not interfere with the propagation of fault exceptions on their way to the barrier
- AOP(Aspect Oriented Programming)
  - ASPECTJ or Spring AOP
  - encapsulating fault handling

Conclusion

- conventions about excetpion
- Thinking of exception of faults and contingencies
- Keep your application simple, maintainable, correct
- AOP may offer some definite advantages.