Discussion-8 exceptions in Java

Discussion Topic:

Handling exceptional conditions can lead to code that can lead to control flow issues and performance degradation within a Java application.
In regard to developing appropriate exceptions in Java, what are some best practices to follow?
How can it be determined when it is optimal to use a user-defined exception with an appropriate try/catch statement?
What security and performance issues should be analyzed?
Be sure to provide an appropriate source code example to illustrate your points.

My Post:

Hello Class,

Handling exceptional conditions is a fundamental aspect of the development of Java applications. However, if not handled properly, they can lead to control flow issues and performance degradation within a Java application. By following the best practices, developers can minimize performance issues and avoid control flow issues, as well as improve the security and maintainability of the application source code.

Best practices such as the SEI CERT Oracle Coding Standard for Java, below is a list of those standards:

- ERR00-J. Do not suppress or ignore checked exceptions
- ERR01-J. Do not allow exceptions to expose sensitive information
- ERR02-J. Prevent exceptions while logging data
- ERR03-J. Restore prior object state on method failure
- ERR04-J. Do not complete abruptly from a finally block
- ERR05-J. Do not let checked exceptions escape from a finally block
- ERR06-J. Do not throw undeclared checked exceptions
- ERR07-J. Do not throw RuntimeException, Exception, or Throwable
- ERR08-J. Do not catch NullPointerException or any of its ancestors
- ERR09-J. Do not allow untrusted code to terminate the JVM

(CMU, n.d.)

Below are some of examples of handling exceptions properly:

- Do not suppress exceptions; in other words, not logging/reporting or re-throwing exceptions, for example, by using an empty catch block. This may result in inconsistent program states. Always handle exceptions by logging them or re-throwing; this adheres to ERR00-J rule "*Do not suppress or ignore checked exceptions*"

Example of compliant code:

```
boolean validFlag = false;
do {
  try {
    // ...
```

```
    // If requested file does not exist, throws FileNotFoundException
    // If requested file exists, sets validFlag to true
    validFlag = true;
  } catch (FileNotFoundException e) {
    // Ask the user for a different file name
  }
} while (validFlag != true);
// Use the file
```

Code from “Rule 07. Exceptional Behavior (ERR). ERR00-J. Do not suppress or ignore checked exceptions” (CMU, n.d.)

- Do not reveal sensitive information in exception messages, such as system paths, sensitive file names, or user credentials. Generic messages should be used and making sure to sanitize exceptions is good practice. This adheres to rule ERR01-J “*Do not allow exceptions to expose sensitive information*”

Example of compliant code:

```
try {
    // Access restricted file
} catch (FileNotFoundException e) {
    throw new SecurityException("Access denied.");
}
```

- It is best practice to avoid using ‘*System.out*’ or ‘*printStackTrace*’ for displaying logging errors. Instead, utilize secure logging frameworks like ‘*java.util.logging*’ or ‘*SLF4J*’. This adheres to rule ERR02-J “*Prevent exceptions while logging data*”.

Example of compliant code:

```
private static final Logger logger = Logger.getLogger(MyClass.class.getName());

try {
    // Perform an operation
} catch (Exception e) {
    logger.log(Level.SEVERE, "Critical failure", e);
}
```

- Standard exceptions often do not cover fully the context of an error. It is best practice when it is possible to handle errors with custom exceptions and or when restoring the application functionality without propagating the error. In other words, in most cases, it is optimal to use a user-defined exception with an appropriate try/catch statement. However, in cases such as ‘*IllegalArgumentException*’, ‘*NullPointerException*’, and ‘*IOException*’ which are self-explanatory exceptions, it can be argued that a custom exception adds little value. In other words, when errors are not domain-specific, that is when errors are not specific to the application's domain or logic.

Example of user-define try/catch block:

```
// Custom exception
public class InsufficientBalanceException extends Exception {
    public InsufficientBalanceException(String message) {
        super(message);
    }
}

// Usage
public void withdraw(double amount) throws InsufficientBalanceException {
    if (amount > balance) {
        throw new InsufficientBalanceException("Insufficient balance.");
```

```
    }
    balance -= amount;
}
```
The code above complies with the ERR00-J and ERR01-J rules.

Another aspect of improperly using try/catch blocks is overuse, Java developers tend to integrate try/catch blocks everywhere in the code when simple tests should suffice. This may add unnecessary overhead and performance issues. Additionally, “the ability to throw an exception in a method is there to take care of really abnormal errors that the method itself can't be expected to handle and that should therefore be handled by the calling method (or its caller ...). Don't use exception handling for other things” (Gries, 2017).

To summarize, if not handled properly, exceptional conditions can lead to code that can lead to control flow issues and performance degradation within a Java application. By adhering to the SEI CERT Oracle Coding Standards and implementing best practices, developers can avoid issues such as overuse of try/catch blocks, improper handling of exceptions, and exposing sensitive information. Custom exceptions are used in domain-specific scenarios, while standard exceptions should suffice for general errors.

-Alex

References:

CMU (n.d). Rule 07. Exceptional Behavior (ERR). *SEI CERT Oracle Coding Standard for Java*. Carnegie Mellon University. Software Engineering Institute. https://wiki.sei.cmu.edu/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java

Gries, D. (2017). 6. Hints on using exceptions*. JavaHyperText and data structures.* Department of Computer Science. Cornell University. https://www.cs.cornell.edu/courses/JavaAndDS/exceptions/EX6.html?utm_source=chatgpt.com