A Philosophy of Software Design

Error out of Existence

Adarsh Pal

7. Mai 2023

Introduction

It is a software design emphasizes the importance of designing systems in a way that minimizes the likelihood of errors occurring in the first place. This approach focuses on preventing errors from happening rather than simply detecting and handling them after they occur.

- The goal is to create software systems that are more reliable, robust, and easier to maintain.
- Errors in software can have significant negative impacts, such as crashes, data loss, or security vulnerabilities.
- In most programming languages,it is coded in try-catch blocks.(Exception Handling)

Here is the example in Python

```
try:
num1 = int(input(Ënter a number: "))
num2 = int(input(Ënter another number: "))
result = num1 / num2
print(The result is:", result)
except ValueError:
print(İnvalid input, please enter a number.")
except ZeroDivisionError:
print(Cannot divide by zero.")
except:
print(Än unknown error occurred.")
```

- The code inside the try block accepts two input values from the user and calculates the result by dividing the first number with the second number.
- If there is a ValueError or ZeroDivisionError, the program jumps to the corresponding except block to handle the error. If an unknown error occurs,the final except block will catch it.
- If the user enters a non-numeric value, the program will jump to the ValueError except block and print the message Ïnvalid input.
- Similarly, if the user enters a 0 for the second number, the program will jump to the ZeroDivisionError except block and print Cannot divide by zero."

Exception Add Complexity

Many programming language include a formal exception mechanism that allows exceptions to be thrown by the lower level code and caught by enclosing code.However,when the new mechanism is added it must me integrated with the existence mechanism or else it will change the flow of program and add more complexity to it.

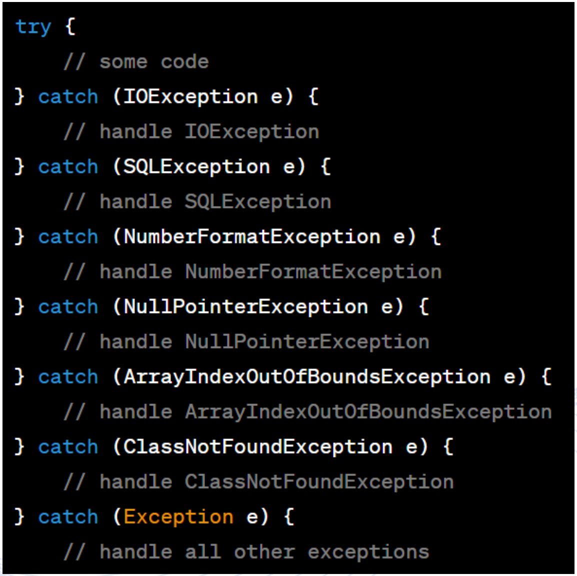
Several different ways a particular piece of code encounter exceptions:

- A caller may provide bad argument or configuration information.
- The code may detect bugs, internal inconsistencies or situations is not prepared to handle.
- In distributed system, network packets may be lost or delayed, servers may not respond in timely fashion.

University of Applied Sciences
HOCHSCHULE
EMDEN·LEER

Too Many Exceptions

- Most Programmer are taught that it's important to detected and reports error; they often interpret this to mean the more errors detected, the better."
- Excessive use of exception can be a problematic, it make code hard to understand. If it is used too many then, it become too difficult to predict its behavior and identify the root cause of error.
- Exception should be used only in case when it is necessary such as dealing with unpredictable error that can be easily handle in normal control flow of program.
- To overcome from this situation is very easy just reduce the number of places were the exception have to be handled.

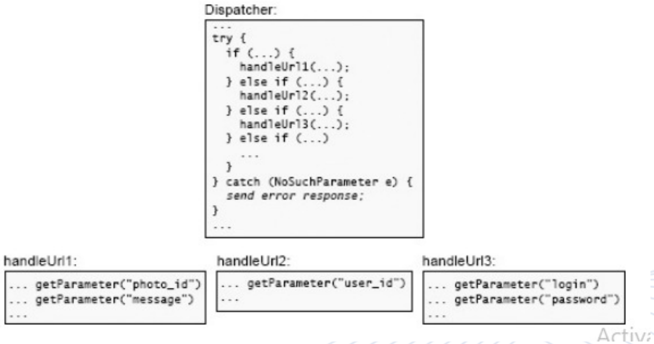


```
try {
    // some code
} catch (IOException e) {
    // handle IOException
} catch (SQLException e) {
    // handle SQLException
} catch (NumberFormatException e) {
    // handle NumberFormatException
} catch (NullPointerException e) {
    // handle NullPointerException
} catch (ArrayIndexOutOfBoundsException e) {
    // handle ArrayIndexOutOfBoundsException
} catch (ClassNotFoundException e) {
    // handle ClassNotFoundException
} catch (Exception e) {
    // handle all other exceptions
```

Abbildung:

Mask Exceptions

- The masking technique involves disabling or blocking the propagation of certain exceptions, allowing the program or system to continue executing without being interrupted by the exception. This can be useful in situations where certain exceptions are expected and can be handled by the program or system without causing any significant issues.
- For example, in a program that handles network communication, a mask exception could be used to prevent the program from raising an exception when a network connection is lost temporarily. Instead of interrupting the program and causing it to terminate, the exception is masked, and the program continues running until the network connection is restored. [Ous18]



Dispatcher:

```
...
try {
  if (...) {
    handleUrl1(...);
  } else if (...) {
    handleUrl2(...);
  } else if (...) {
    handleUrl3(...);
  } else if (...)
    ...
  }
} catch (NoSuchParameter e) {
  send error response;
}
...
```

handleUrl1:

```
... getParameter("photo_id")
... getParameter("message")
...
```

handleUrl2:

```
... getParameter("user_id")
...
```

handleUrl3:

```
... getParameter("login")
... getParameter("password")
...
```

Abbildung: **Example [Ous18]**

Aggregation Exceptions

- Exception aggregation is the process of collecting multiple related exceptions and consolidating them into a single exception object. This can be useful in situations where multiple errors occur within a system or application, and it is desirable to present the user with a single error message or log a single exception for easier troubleshooting.
- It can also be useful in distributed systems, where errors may occur in different components of the system and need to be consolidated into a single error message or exception for easier management and debugging.
- It should not be used to mask or hide underlying errors. Rather, it should be used as a tool for better presentation and management of errors and exceptions.

Just Crash

- It is unexpected and sudden failure of a software program or system. It occurs when the program encounters an error or a condition that it cannot handle, leading to the program being terminated or crashing."When a crash occurs, the program may stop functioning, display an error message, or cause other problems, such as data loss or corruption.
- Crashes can occur due to a variety of reasons, including bugs or errors in the program's code, insufficient system resources, hardware malfunctions, or conflicts with other software programs or components. They can also occur due to unexpected inputs or user actions that the program is not designed to handle.
- To prevent crashes, software developers use various techniques, such as testing, debugging, and error handling, to identify and address potential issues before they occur.

Design Special

- It allows software developers to create software that is tailored to the unique needs and requirements of specific industries and domains. This type of software can help to improve efficiency, accuracy, and productivity, and can have a significant impact on the success of businesses and organizations.
- Designing special software requires a collaborative effort between software developers and domain experts. Developers must work closely with subject matter experts to understand the specific needs and requirements of the target users and to ensure that the software meets their needs.

Example: Simulation tools and visualization and data form for a organizations

Taking it too Far

- When software developers over-design or over-build a solution beyond the actual requirements, leading to unnecessarily complex software that is difficult to maintain and update. [Tor18]

  Consequence are as below:-
- Increased development time and cost
- Reduced maintainability and usability
- Reduced performance

- Developers should prioritize simplicity and ease of use, avoiding unnecessary complexity or features that do not add value.

University of Applied Sciences
HOCHSCHULE
EMDEN·LEER

Conclusion

Errors or bugs should be eliminated as soon as they are discovered.The longer a bug remains in the code, the more difficult and expensive it becomes to fix.Developers should prioritize testing and debugging throughout the development process. Along with use tools and techniques such as automated testing and code review to identify errors and bugs. Software developers can create more reliable and efficient software that meets the needs of their users.

Thank You

For your attention

University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Reference

References I

[Ous18] John Ousterhout. *A Philosophy of Software Design*. Boston, Massachusetts: Addison-Wesley Professional, 2018. ISBN: 978-0134848717.

[Tor18] Adam Tornhill. *Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis*. Raleigh, North Carolina: Pragmatic Bookshelf, 2018. ISBN: 978-1-68050-286-1.