Error Handling Use Exceptions Rather Than Return Codes

This part talks about how we've improved the way we deal with mistakes in programming. In the past, when errors happened, it made the code messy because we had to constantly check for them after each action. This made it easy to miss things. But now, by using exceptions, we make the code neater and easier to understand. Errors are handled separately from the main part of the code, which makes it clearer and easier to read. This helps developers concentrate on different parts of the program without getting confused, which makes the code better and easier to manage.

Write Your Try-Catch-Finally Statement First

The paragraph talks about how important it is to handle exceptions in computer programs. It compares using try-catch-finally statements to making sure everything in a program is done correctly, like in a transaction. It's saying that by using these structures early on when writing code, it helps users know what to expect and makes it easier to deal with mistakes. It also mentions that testing and examples show how vital it is to think about and deal with exceptions properly, stressing the need to make sure code runs smoothly even if there are errors.

Use Unchecked Exceptions

The passage talks about the discussion on checked exceptions in Java. It says that although they were thought to be helpful at first, now people feel they aren't needed for making reliable software. It explains that checked exceptions break a rule called the Open/Closed Principle, leading to big changes in the code whenever exceptions are used. This can mess up how the code is organized and make it harder to manage. So, it suggests making exception classes based on what the caller needs instead of using them everywhere in the code.

Provide Context with Exceptions

In Java programming, it's important to have clear error messages when something goes wrong. Just seeing a stack trace isn't enough to know what went awry. By making helpful error messages and sending them with exceptions, developers can explain exactly what went wrong and why. This doesn't just help find and fix bugs but also makes it easier to keep track of errors in the program.

Define Exception Classes in Terms of a Caller’s Needs

This part talks about dealing with mistakes and sorting them into groups. It says it's more important how we catch errors than how we label them. There's an example of mixing up error types and a solution: making all errors look the same by wrapping them up. Doing this makes the code simpler, cuts down on relying too much on other things, makes testing easier, and lets us design APIs more flexibly. It stresses the idea of using one type of error as much as we can and only using different types when we really need to deal with them differently.

Define the Normal Flow

This part emphasizes the idea of keeping the main rules of a business separate from how errors are dealt with. This helps keep the code neat and easy to manage. By handling errors at the start and end of the program and using methods like wrapping external systems and defining handlers, the code can be made more straightforward. But sometimes, especially when dealing with special situations like expenses in a billing program, this method might make the code messy. The suggested fix is to use the Special Case Pattern, where a specific class or object is set up to deal with these special cases, making it simpler for the main code and neatly wrapping up any unusual behavior.

Don’t Return Null

The passage talks about why using "null" in code is not good and how it can make mistakes more likely and code harder to understand. Instead of using "null," it's better to use other methods like throwing exceptions or returning special objects. For example, instead of returning nothing, developers can use a special object like Collections.emptyList() to make their code cleaner and less likely to cause errors.

Don’t Pass Null

This part talks about why using "null" in code can cause problems. It says it's better to avoid using it to make code clearer and less likely to have errors. Instead of using "null," it recommends using other methods like throwing exceptions or using special objects. For example, instead of returning "null," developers can use a special object called Collections.emptyList(). This helps make the code cleaner and less likely to have certain types of errors.