1Z0-808 Exam Topic Reviewer

TopicId: 1030
Topic: Throwing and Creating Exceptions

August 5, 2025

Taking Control: Throwing and Creating Exceptions

Alright team, we've learned how to catch exceptions that the Java API throws at us. Now it's time to take control. Sometimes, your own application logic will encounter an error state that cannot be resolved within the current method. In these cases, your method needs to signal this problem to the caller. We do this by explicitly creating and throwing an exception object.

0.1 `throw` vs. `throws`: The Ultimate Showdown

This is one of the most confused pairs of keywords in Java. The exam will absolutely test your understanding of their different roles. Do not mix them up.

Feature	`throw`	`throws`
Purpose	An **action** to programmatically throw an exception.	A **declaration** in a method
Usage	Inside a method body.	As part of the method signa
Argument	A single exception *instance* (object).	One or more exception *class*
Example	`throw new IOException();`	`void method() throws IOE`

Let's see them in action:

```
// The 'throws' keyword is a warning to the caller.
public void processFile(String fileName) throws FileNotFoundException {
    if (fileName == null) {
        // The 'throw' keyword is the action of throwing the exception.
        throw new FileNotFoundException("File name cannot be null!");
    }
    // ... more code
}
```

The `throws` clause here is necessary because `FileNotFoundException` is a *checked* exception. This forces any method that calls `processFile()` to either handle it with a `try-catch` block or declare it in its own `throws` clause.

0.2 Creating Your Own Exceptions

Sometimes, built-in exceptions like `IllegalArgumentException` aren't specific enough. You can create your own custom exception classes to make your code more readable and your error conditions more precise.

- **To create a CHECKED exception**, extend `java.lang.Exception`.

```
public class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}
```

- **To create an UNCHECKED exception**, extend `java.lang.RuntimeException`.

```
public class InvalidTransactionIdException extends RuntimeException {
    public InvalidTransactionIdException(String message) {
```

```
            super(message);
        }
    }
```

The choice of parent class is critical. If you extend `Exception`, the compiler will enforce the "handle or declare" rule on any method that throws your custom exception. If you extend `RuntimeException`, it becomes an unchecked exception, and the compiler will not force the caller to handle it.

Key Takeaways for the 1Z0-808 Exam

- `throw` is an action inside a method that throws a single exception *object*.
- `throws` is a declaration in a method signature that lists the checked exception *types* it might throw.
- You can only throw objects that are instances of `Throwable` or one of its subclasses.
- When creating custom exceptions, extend `Exception` for checked exceptions and `RuntimeException` for unchecked exceptions.