Was passiert nach dem `throw`

C++ sieht nach, ob das `throw` von einem Errorhandler eingerahmt ist, der das geworfene Objekt auffangen kann. Das Einrahmen erfolgt durch **`try`** `{` am Beginn und `}` **`catch`**`(exception-definition) {…}` am Ende. Die Punkte stehen für den Code, der im Fehlerfall ausgeführt werden soll. Man nennt diese Konstruktion wenig überraschend einen `try-catch`-Block:

```
try {
     …
     throw 5;
}
catch(int e) {
      cout << "Fehler Nr. " << e << " aufgetreten\n"; }
catch(const char* text) {
      cout << "Fehler \"" << text << "\" aufgetreten\n"; }
catch(...) {
      cout << "unbekannter Fehler aufgetreten\n"; }
```

Man sieht, dass man mehrere Handler mit eigenem Code angeben kann. Diese werden der Reihe nach dahingehend überprüft, ob der Typ, den sie auffangen können, mit dem geworfenen Objekt kompatibel ist. Der erste anwendbare Handler wird ausgeführt und alle folgenden werden ignoriert. Die letzte `catch(...)` Bedingung oben ist die „fange alles“-Phrase, sie muss, falls sie verwendet wird, immer am Ende aller Handler stehen.

Kann keiner der Handler angewendet werden, geschieht dasselbe, was auch bei einem fehlenden `try-catch`-Block geschehen würde: Die rekursive Abarbeitung (Stack Unwinding)**:**

C++ führt an der Fehlerstelle ein sofortiges `return` (ohne Rückgabewert) aus und sucht beim Aufrufer der fehlerhaften Funktion an der Aufrufstelle nach einem passenden Handler. Wird ein solcher gefunden, kommt er zum Zug. Falls auch dort keiner existiert, macht C++ wieder ein `return` und sucht eine Ebene tiefer weiter usw. Gibt es nirgendwo einen passenden Handler, fällt C++ zuletzt aus der `main()`−Funktion heraus und das Programm endet mit einer „unhandled exception ... program terminated“ oder ähnlichen Fehlermeldungen.

Bemerkung: Durch das ausgeführte `return` im Exception-Fall werden alle Destruktoren der lokalen Objekte ausgeführt, während der Rest der Funktion übersprungen wird. Man sollte daher **wichtige kritische** Aufräumarbeiten eher in einem Destruktor als durch normalen Code ausführen, weil ersterer auch im Exceptionfall ausgeführt wird!