Try/finally compilation issues

Nashorn incorrectly compiles try/finally constructs in a variety of cases. The problems exist with all released Java 8 versions, up to and including 8u40. They are fixed in both 8u60 and 9 early release builds.

Double finally execution with break and continue

In one case, a finally block will execute before a break or continue statement inside a try/finally. Consider the program below:

```javascript
try {
    for(var i = 0; i < 3; ++i) {
        print("i=" + i);
        if(i == 1) {
            print("continue");
            continue;
        }
    }
} finally {
    print("finally");
}
```

It unfortunately prints:

```
i=0
i=1
continue
finally
i=2
finally
```

Both a continue and a break statement inside a try/finally block will trigger immediate execution of the finally block (that is, the program above would run the finally block twice if you replaced continue with break). This is caused by a fault in the compiler logic that ensures that finally blocks are executed before the jump when a continue or break jump outside the try block. Unfortunately, due to the bug, the compiler will emit code that executes the finally block at the jump point even when the jump is within the try block.

Workaround

The compiler stops analysis at function boundaries, so by utilizing an immediately-invoked function expression, you can neutralize the bug:

```javascript
try {
    (function() {
        for(var i = 0; i < 3; ++i) {
            print("i=" + i);
            if(i == 1) {
                print("continue");
                continue;
            }
        }
    })();
} finally {
    print("finally");
}
```

Of course, this only works if you don't have a control flow transfer to outside the try block in its body (e.g. a return statement).

Catches catch exceptions thrown in their sibling finally blocks

If a finally block in a try/catch/finally throws an exception, it can get caught by its sibling catch block if the try block contains a transfer of control outside of itself (a break, continue, or return). Let's consider the below program:

```javascript
(function() {
    try {
        return;
    } catch(e) {
        print("Caught " + e);
    } finally {
        print("Finally");
        throw "finally-thrown";
    }
})();
```
It prints:

```javascript
Finally
Caught finally-thrown
Finally
test.js:8:8 finally-thrown
```

Showing that the catch block has caught the exception thrown from the finally block. There is no known workaround for this problem.