The try-catch Statement to Catch Exceptions

This lesson explains how we can use try-catch statements to catch exceptions.

WE'LL COVER THE FOLLOWING

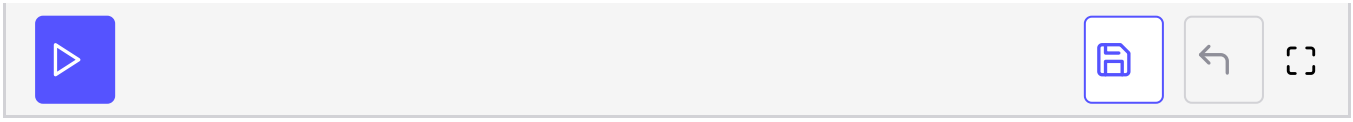
The try-catch statement to catch exceptions

As we've seen earlier in this chapter, a thrown exception causes the program execution to exit all functions, and this finally terminates the whole program. The exception object can be caught by a try-catch statement at any point on its path as it exits the functions. The try-catch statement models the phrase "try to do something, and catch exceptions that may be thrown." Here is the syntax of try-catch:

```
try {
    // the code block that is being executed, where an exception may be th
rown
} catch (an_exception_type) {
    // expressions to execute if an exception of this type is caught
} catch (another_exception_type) {
    // expressions to execute if an exception of this other type is caught

// ... more catch blocks as appropriate ...
} finally {
    // expressions to execute regardless of whether an exception is thrown
}
```

Let's start with the following program, which does not use a try-catch statement at this stage. The program reads the value of a die from a file and writes its value to the standard output:

```
// this program is expected to fail execution

import std.stdio;

int readDieFromFile() {
    auto file = File("the_file_that_contains_the_value", "r");

    int die;
    file.readf(" %s", &die);

    return die;
}

void main() {
    const int die = readDieFromFile();

    writeln("Die value: ", die);
}
```



Program without try-catch statement

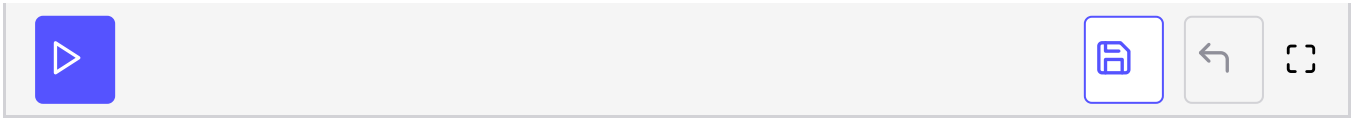
Note that the `readDieFromFile` function is written in a way that ignores error conditions, assuming that the file and the value that it contains are available. In other words, the function is dealing only with its own task instead of paying attention to error conditions. This is a benefit of using exception handling: many functions can be written in ways that focus on their actual tasks, rather than focusing on error conditions. Let's start the program when `the_file_that_contains_the_value` is missing:

```
std.exception.ErrnoException@std/stdio.d(286): Cannot open file `the_file_
that_contains_the_value' in mode `r' (No such file or directory)
```

An exception of type `ErrnoException` is thrown and the program terminates without printing `Die value:` . Let's add an intermediate function to the program that calls `readDieFromFile` from within a `try` block, and let's have `main()` call this new function:

```
import std.stdio;

int readDieFromFile() {
    auto file = File("the_file_that_contains_the_value", "r");

    int die;
```

```
    file.readf(" %s", &die);

    return die;
}

int tryReadingFromFile() {
    int die;

    try {
        die = readDieFromFile();

    } catch (std.exception.ErrnoException exc) {
        writeln("(Could not read from file; assuming 1)");
        die = 1;
    }

    return die;
}

void main() {
    const int die = tryReadingFromFile();

    writeln("Die value: ", die);
}
```



Program with try-catch statement

We start the program again when `the_file_that_contains_the_value` is still missing; this time the program does not terminate with an exception:

```
(Could not read from file; assuming 1)
Die value: 1
```

The new program tries executing `readDieFromFile` in a `try` block. If that block executes successfully, the function ends normally with the `return die;` statement. If the execution of the `try` block ends with the specified `std.exception.ErrnoException`, then the program execution enters the `catch` block.

The following is a summary of events that take place when the program is started with a missing file:

- a `std.exception.ErrnoException` object is thrown (by `File()`, not by our code)
- the exception is caught by `catch`

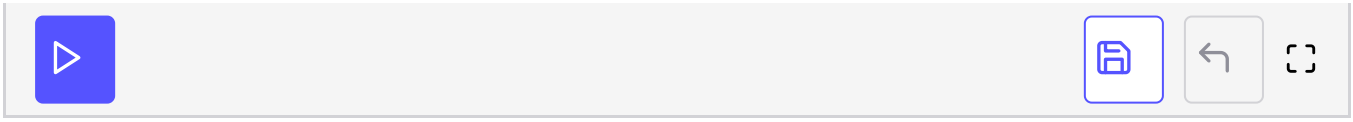
- the die is assumed to have a value of 1 during the normal execution of the catch block
- the program continues its normal operations.

`catch` is to catch thrown exceptions in order to find a way to continue executing the program. As another example, let's go back to the omelet program and add a try-catch statement to its `main()` function:

```
import std.stdio;
import std.stdio;

void indent(int level) {
    foreach (i; 0 .. level * 2) {
        write(' ');
    }
}

void entering(string functionName, int level) {
    indent(level);
    writeln("▶ ", functionName, "'s first line");
}

void exiting(string functionName, int level) {
    indent(level);
    writeln("◁ ", functionName, "'s last line");
}

void RENAMED_main() {
    entering("main", 0);
    makeOmelet();
    eatOmelet();
    exiting("main", 0);
}

void makeOmelet() {
    entering("makeOmelet", 1);
    prepareAll();
    cookEggs();
    cleanAll();
    exiting("makeOmelet", 1);
}

void eatOmelet() {
    entering("eatOmelet", 1);
    exiting("eatOmelet", 1);
}

void prepareAll() {
    entering("prepareAll", 2);
    prepareEggs();
    prepareButter();
    preparePan();
    exiting("prepareAll", 2);
}

void cookEggs() {
    entering("cookEggs", 2);
```

```
    exiting("cookEggs", 2);
}

void cleanAll() {
    entering("cleanAll", 2);
    exiting("cleanAll", 2);
}

void prepareEggs() {
    entering("prepareEggs", 3);
    exiting("prepareEggs", 3);
}

void prepareButter() {
    entering("prepareButter", 3);
    exiting("prepareButter", 3);
}

void preparePan() {
    entering("preparePan", 3);
    exiting("preparePan", 3);
}
import std.string;

// ...

void prepareEggs(int count) {
    entering("prepareEggs", 3);

    if (count < 1) {
        throw new Exception(
            format("Cannot take %s eggs from the fridge", count));
    }

    exiting("prepareEggs", 3);
}
// ...

void RENAMED_main() {
    entering("main", 0);
    makeOmelet(-8);
    eatOmelet();
    exiting("main", 0);
}

void makeOmelet(int eggCount) {
    entering("makeOmelet", 1);
    prepareAll(eggCount);
    cookEggs();
    cleanAll();
    exiting("makeOmelet", 1);
}

// ...

void prepareAll(int eggCount) {
    entering("prepareAll", 2);
    prepareEggs(eggCount);
    prepareButter();
    preparePan();
    exiting("prepareAll", 2);
}
```

```
// ...
void main() {
    entering("main", 0);

    try {
        makeOmelet(-8);
        eatOmelet();

    } catch (Exception exc) {
        write("Failed to eat omelet: ");
        writeln('"', exc.msg, '"');
        writeln("Will eat at neighbor's...");
    }

    exiting("main", 0);
}
```



Omelet program with try-catch statement

> **Note:** The `.msg` property will be explained below.

That `try` block contains two lines (118,119) of code. Any exception thrown from either of those lines would be caught by the `catch` block.

```
▶ main, first line
  ▶ makeOmelet, first line
    ▶ prepareAll, first line
      ▶ prepareEggs, first line
Failed to eat omelet: "Cannot take -8 eggs from the fridge" Will eat at ne
ighbor's...
◁ main, last line
```

As seen from the output, the program doesn't terminate because of the thrown exception anymore. It recovers from the error condition and continues executing normally until the end of the `main()` function.

Order of `catch` blocks

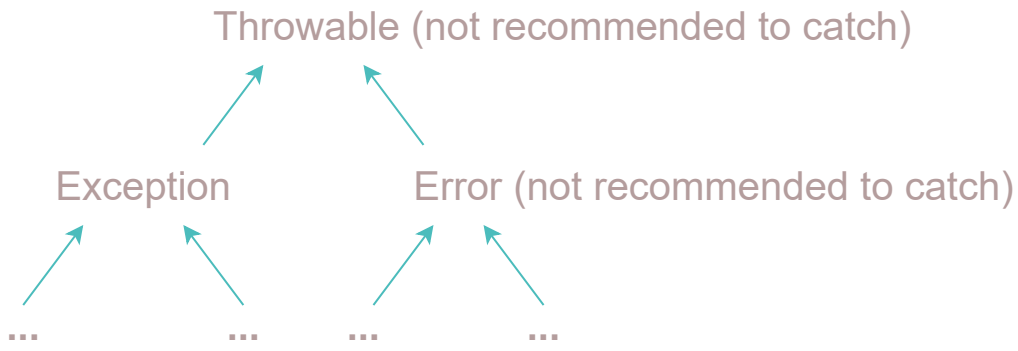
The `catch` blocks are executed sequentially. The type `Exception`, which we have used so far in the examples is a general exception type. This type merely specifies that an error occurred in the program. The type also contains a message that can explain the error further, but it does not contain

information about the type of error.

`ConvException` and `ErrnoException`, which we have seen earlier in this chapter are more specific exception types: the former is about a conversion error, and the latter is about a system error. Like many other exception types in Phobos and as their names suggest, `ConvException` and `ErrnoException` are both inherited from the `Exception` class.

`Exception` and its sibling `Error` are further inherited from `Throwable`, the most general exception type.

Although possible, it is not recommended to catch objects of type `Error` and objects of types that are inherited from `Error`. Since it is more general than `Error`, it is not recommended to catch Throwable either. What should normally be caught are the types that are under the `Exception` hierarchy, including `Exception` itself.



> **Note:** The tree above indicates that Throwable is the most general and Exception and Error are more specific.

It is possible to catch exception objects of a particular type. For example, it is possible to catch an `ErrnoException` object specifically to detect and handle a system error.

Exceptions are caught only if they match a type that is specified in a catch block. For example, a catch block that is trying to catch a `SpecialExceptionType` would not catch an `ErrnoException`.

Working of try-catch blocks

The type of the exception object that is thrown during the execution of a try block is matched to the types that are specified by the catch blocks in the

order in which the catch blocks are written. If the type of the object matches

the type of the catch block, then the exception is caught by that catch block, and the code within this catch block is executed. Once a match is found, the remaining catch blocks are ignored.

Because the catch blocks are matched in order from the first to the last, the catch blocks must be ordered from the most specific exception types to the most general exception types. Accordingly, if it is required, the `Exception` type must be specified at the last catch block.

For example, a try-catch statement that is trying to catch several specific types of exceptions about student records must order the catch blocks from the most specific to the most general as in the following code:

```
try {
    // operations about student records that may throw ...

} catch (StudentIdDigitException exc) {
    // an exception that is specifically about errors with the digits of s
tudent ids

} catch (StudentIdException exc) {
    // a more general exception about student ids but not necessarily abou
t their digits

} catch (StudentRecordException exc) {
    // even more general exception about student records

} catch (Exception exc) {
    // the most general exception that may not be related to student recor
ds
}
```

In the next lesson, we will see the finally block of the try-catch statement.