Answer the discussion questions for chapter 6 in writing.

1. What is an exception?

An exception replaces the traditional techniques of manually adding error-detecting code around every statement. Rather, with an exception you write your code in a try block, and if an error occurs, the runtime throws an exception, meaning that the runtime then examines the catch handlers for the different errors that could arise in the try block, and then execution jumps out of the try block and into the pertinent catch handler.

1. What happens in a try block if the program executes without errors?

When the code runs, it attempts to execute all the statements in the try block, and if none of the statements generates an exception, they all run, one after the other, to completion.

1. How does the catch mechanism work for unhandled exceptions?

If unable to find a matching catch handler, the program will terminate with an unhandled exception. Although in the .NET Framework, the Exception handler traps every possible exception that can occur.

1. What happens in a program if an exception block fails to handle a particular error?

If the try block is part of a method, the method immediately exits and execution returns to the calling method.

If the calling method uses a try block, the runtime attempts to locate and execute a matching catch handler for this try block

If the calling method does not use a try block or if there is no matching catch handler, the calling method exits, and execution returns to its caller, where the process is repeated.

If a matching catch handler is found, the handler runs and execution continues with the first statement that follows the catch handler in the catching method.

1. What is the parent class for all exceptions? How does this work?

Exception is the parent class for all exceptions. Exceptions are organized into families called inheritance hierarchies. For example, FormatException and OverflowException both belong to a family called SystemException. You should place more specific catch handlers above a general catch handler.

1. How do you determine the type of an error?

Some exceptions are the result of other exceptions raised earlier. You can determine the type of error by expanding the InnerException property in the View detail dialog box and find the exception with property set to null.

1. What is the purpose of integer checking?

An int in C# is 32 bits, and the exact range of values is -2147483648 to 2147483647. Without integer checking, if 1 is added to the int.MaxValue, the caculation wraps around to the int.MinValue. Integer checking allows an OverflowException to be thrown instead.

1. What does the finally block do?158071679902/03/202002:59am

* [File Upload](https://erau.instructure.com/courses/107848/assignments/1921777?module_item_id=5959559#submit_online_upload_form)
* [Text Entry](https://erau.instructure.com/courses/107848/assignments/1921777?module_item_id=5959559#submit_online_text_entry_form)
* [Google Doc](https://erau.instructure.com/courses/107848/assignments/1921777?module_item_id=5959559#submit_google_doc_form)
* [Studio](https://erau.instructure.com/courses/107848/assignments/1921777?module_item_id=5959559#submit_from_external_tool_form_62230)

Top of Form



The finally block ensures that a statement runs regardless of whether or not an exception is thrown. For example, when you open a file for reading, you must ensure that you call reader.Dispose to release the resources. If not, you’ll run out of file handles to open more files. The Finally block can ensure that reader.Dispose executes even if an exception is thrown.