

# The `std::exception` Solutions Exercises

- Explain why the `what()` member function of `std::exception` is virtual
  - A virtual `what()` function allows child classes to override it
  - A child class can return an appropriate error message for the exception it represents
- Explain why the destructor of `std::exception` is virtual
  - In order that dynamic binding is used when destroying child classes
  - This ensures that objects are correctly destroyed

- The child classes `logic_error` and `runtime_error` and their subclasses have a constructor that takes a string argument
- What is this string used for?
  - It populates the error message that is returned by `what()`
- What should we put in this string?
  - Information about the error condition for which the exception is being thrown

- Give some examples of C++ standard library functions throwing subclasses of `std::exception`
  - `bad_alloc` when memory allocation fails
  - `bad_cast` when a dynamic cast fails
  - `out_of_range` when `vector::at()` is given an out-of-bounds index
  - `invalid_argument` when `stoi()` is called with a string that cannot be converted to a number

- There are many situations where the standard library could usefully throw these subclass exceptions but does not.  
Why is this?
  - Exceptions require extra code (to check for exceptions and to handle them) and would add too much overhead for a general purpose library that has to be efficient in all situations
- Why is it useful to learn about these subclass exceptions?
  - They may be used in third-party code
  - We can use them in our own code