Here we have an example of the "Singleton pattern"[2] being used to create the shared `Logger` instance and provide access to it through the `getLogger()` function. The **`static`** local instance of `Logger`, `localLogger`, will be initialized exactly once and then destroyed after normal program termination. In C++03, it would not be safe to call this function concurrently from multiple threads. Conversely, C++11 *guarantees* that the initialization of `localLogger` will happen exactly once even when multiple threads call `getLogger` concurrently.

Multithreaded contexts

The C++11 Standard Library provides several utilities and abstractions related to multithreading. The `std::thread` class is a portable wrapper for a platform-specific thread handle provided by the operating system. When constructing an `std::thread` object with a **`callable object`**, a new thread invoking that `callable object` will be spawned. Prior to destroying such `std::thread` objects, invoking the `join` member function on the thread object is necessary ~~and will block until the background thread of execution completes invoking its callable object~~.

This threading facility from the Standard Library can be used with our earlier example in *Logger example* on page 69 to concurrently attempt to access the `getLogger` function:

```
#include <thread>  // std::thread

void useLogger() { getLogger() << "example"; }  // concurrently called function

int main()
{
    std::thread t0(&useLogger);
    std::thread t1(&useLogger);
        // Spawn two new threads, each of which invokes useLogger.

    // ...

    t0.join();  // Wait for t0 to complete execution.
    t1.join();  // Wait for t1 to complete execution.

    return 0;
}
```

Such use prior to the C++11 thread-safety guarantees, with pre-C++11 threading libraries, could have led to a `data race` during the initialization of `localLogger`, which was defined as a local **`static`** object in `getLogger`. This **`undefined behavior`** might have resulted in invoking the constructor of `localLogger` multiple times, returning from `localLogger` before

[2] **gamma95**, Chapter 3, section "Singleton," pp. 127–134