tains data, constructors, and pure virtual functions.[2] Such inheritance, known as **implementation inheritance**, is decidedly distinct from pure **interface inheritance**, which is often the preferred design pattern in practice.[3] As an example, consider a ~~base~~ class, `NetworkDataStream`, that allows overriding its virtual functions for processing a stream of data from an expanding variety of arbitrary sources over the network:

```
class NetworkDataStream
{
private:
    // ...                   (member data)

public:
    explicit NetworkDataStream(TCPConnection* tcpConnection);
    explicit NetworkDataStream(UDPConnection* udpConnection);
    explicit NetworkDataStream(RawDataStreamHandle* rawDataStreamHandle);

    virtual ~NetworkDataStream();

    virtual void onPacketReceived(DataPacket& dataPacket) = 0;
        // Derived classes must override this method.
};
```

The `NetworkDataStream` class above provides three constructors, with more under development, that can be used assuming no per-packet processing is required. Now, imagine the need for logging information about received packets (e.g., for auditing purposes). Inheriting constructors make deriving from `NetworkDataStream` and overriding (see Section 1.1. “**`override`**” on page 104) `onPacketReceived(DataPacket&)` more convenient because we don’t need to reimplement each of the constructors, which are anticipated to increase in number over time:

```
class LoggedNetworkDataStream : public NetworkDataStream
{
public:
    using NetworkDataStream::NetworkDataStream;

    void onPacketReceived(DataPacket& dataPacket) override
    {
        LOG_TRACE << "Received packet " << dataPacket;    // local log facility
        NetworkDataStream::onPacketReceived(dataPacket);  // Delegate to base.
    }
};
```

Implementing a strong `typedef`

Classic **`typedef`** declarations — just like C++11 **`using`** declarations (see Section 1.1.“**`using`** Aliases” on page 133) — are just synonyms; they offer absolutely no additional type safety

[2] A discussion of this topic is planned for **lakos2a**, section 4.7.

[3] A discussion of this topic is planned for **lakos2b**, section 4.6.