```
struct S  // Object
{
    S(int)  : S(true) { }  // delegating constructor
    S(bool) : S(0)    { }  // delegating constructor
};
```

Suboptimal factoring

The need for delegating constructors might result from initially suboptimal factoring — e.g., in the case where the same **value** is being presented in different forms to a variety of different **mechanisms**. For example, consider the `IPV4Host` class in *Use Cases — Avoiding code duplication among constructors* on page 48. While having two constructors to initialize the host might be appropriate, if either (1) the number of ways of expressing the same value increases or (2) the number of consumers of that value increases, we might be well advised to create a separate value-semantic type, e.g., `IPV4Endpoint`, to represent that value[3]:

```
#include <cstdint>  // std::uint16_t, std::uint32_t
#include <string>   // std::string


class IPV4Endpoint
{
    std::uint32_t d_address;
    std::uint16_t d_port;

public:
    IPV4Endpoint(std::uint32_t address, std::uint16_t port)
        : d_address{address}, d_port{port}
    {
    }

    IPV4Endpoint(const std::string& ep)
        : IPV4Endpoint{extractAddress(ep), extractPort(ep)}
    {
    }
};
```

Note that `IPV4Endpoint` itself makes use of delegating constructors but as a purely private, encapsulated implementation detail. With the introduction of `IPV4Endpoint` into the codebase, `IPV4Host` (and similar components requiring an `IPV4Endpoint` value) can be redefined to have a single constructor (or other previously overloaded member function) taking an `IPV4Endpoint` object as an argument.

[3]The notion that each component in a subsystem ideally performs one focused function well is sometimes referred to as separation of logical concerns or fine-grained, physical factoring; see **dijkstra82** and see **lakos20**, section 0.4, "Hierarchically Reusable Software," section 3.2.7, "Not Just Minimal, Primitive: The Utility `struct`," and section 3.5.9, "Factoring," pp. 20–28, 529–530, and 674–676, respectively.