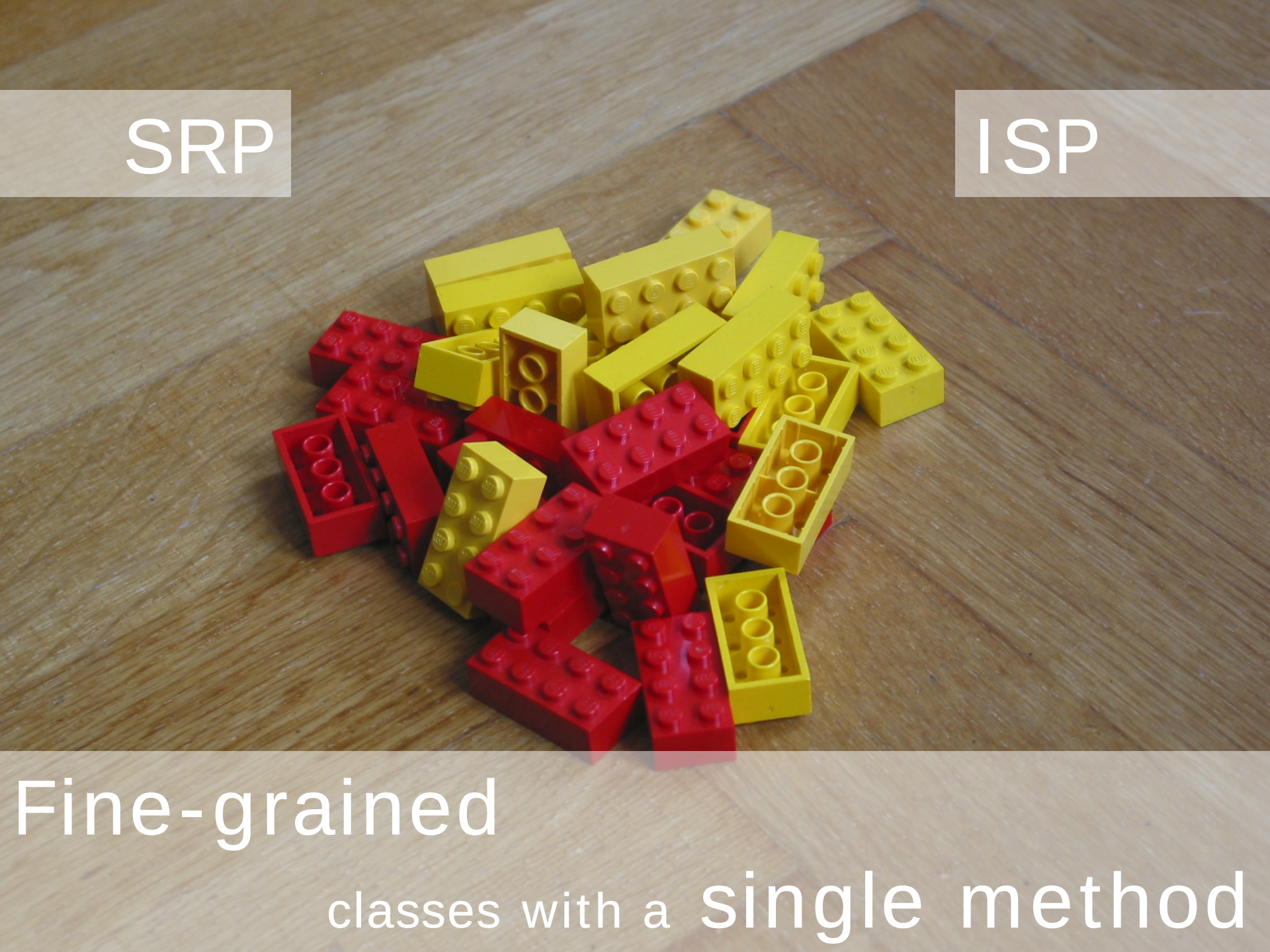
Next step: Functional

The novice


SRP
ISP
Fine-grained
classes with a single method


Objects are data with behaviour

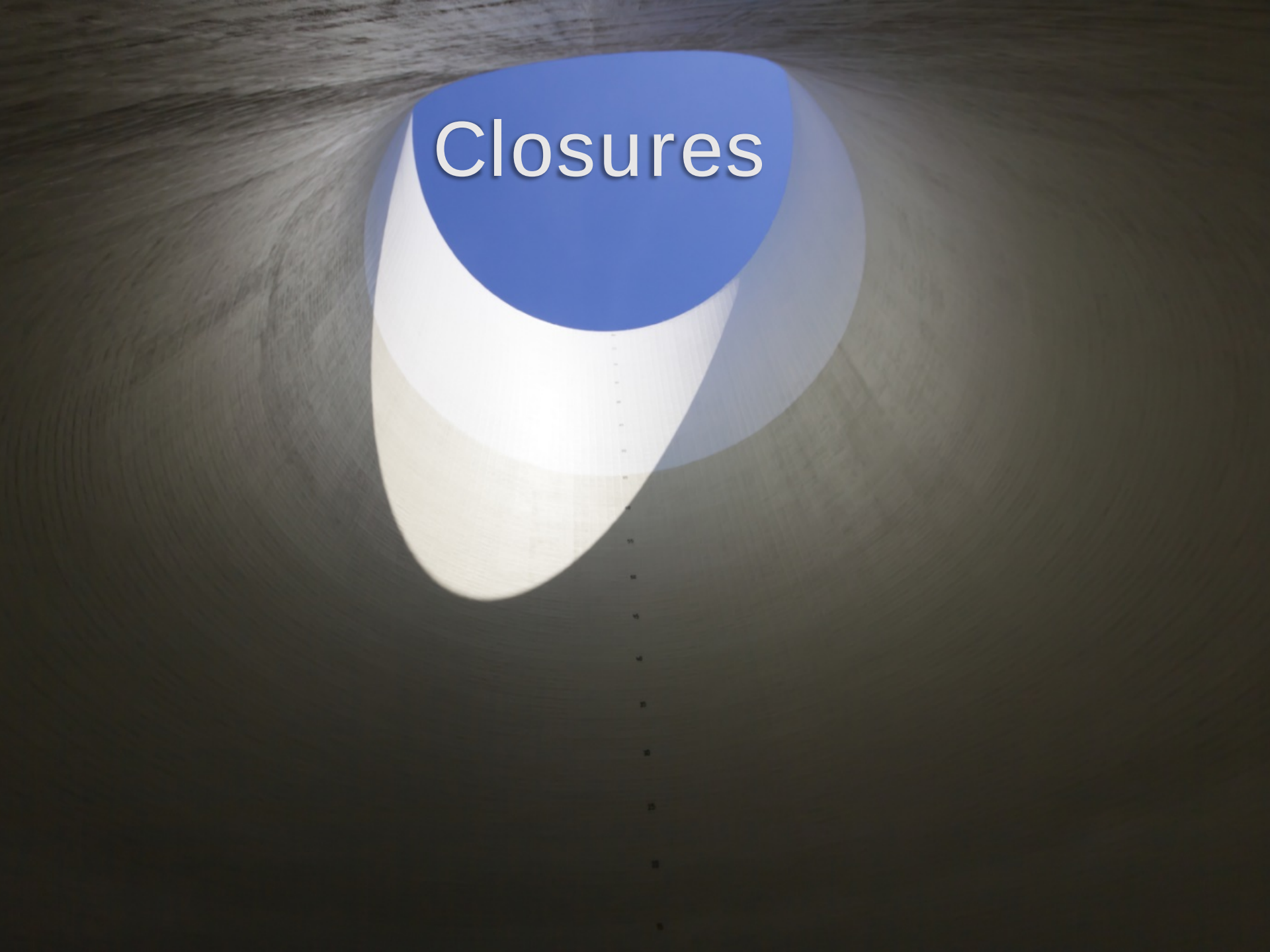
```
public class FileStore : IMessageQuery
{
    private readonly DirectoryInfo workingDirectory;

    public FileStore(DirectoryInfo workingDirectory)
    {
        this.workingDirectory = workingDirectory;
    }

    public string Read(int id)
    {
        var path = Path.Combine(
            this.workingDirectory.FullName,
            id + ".txt");
        return File.ReadAllText(path);
    }
}
```


Functions are pure behaviour

```
Func<DirectoryInfo, int, string> read = (workingDirectory, id) =>
    {
        var path = Path.Combine(
            workingDirectory.FullName,
            id + ".txt");
        return File.ReadAllText(path);
    };
```


Closures

```
var workingDirectory =
    new DirectoryInfo(Environment.CurrentDirectory);



Func<int, string> read = id =>
    {
        var path = Path.Combine(
            workingDirectory.FullName,
            id + ".txt");
        return File.ReadAllText(path);
    };
```


What does that compile to?

```
[CompilerGenerated]
private sealed class <>c__DisplayClass3
{
    public DirectoryInfo workingDirectory;

    public string <UseClosure>b__2(int id)
    {
        return File.ReadAllText(
            Path.Combine(
                this.workingDirectory.FullName,
                id + ".txt"));
    }
}
```


Closures are behaviour with data

What's more lightweight?

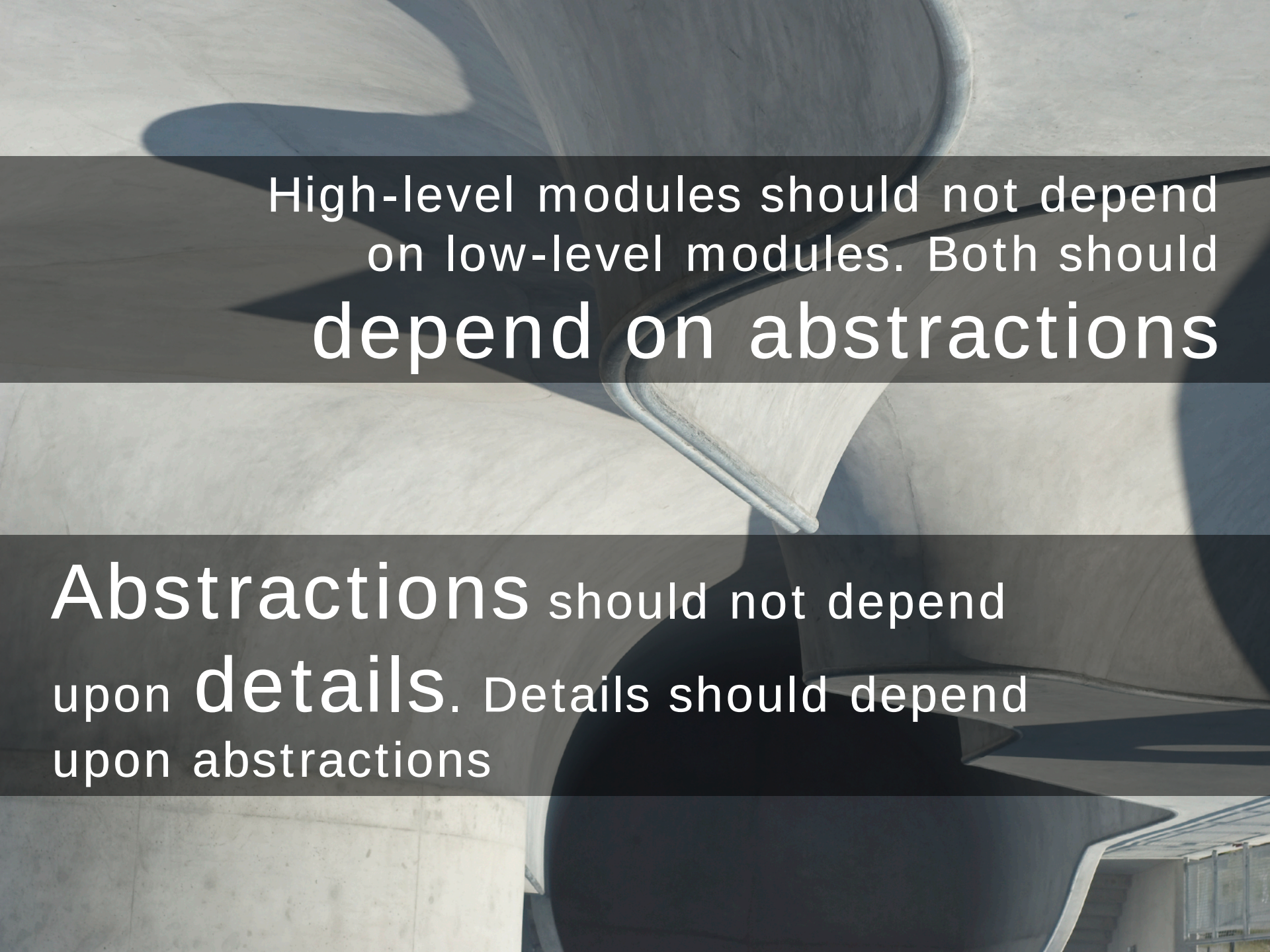
```
var workingDirectory =
    new DirectoryInfo(Environment.CurrentDirectory);

Func<int, string> read = id =>
    {
        var path = Path.Combine(
            workingDirectory.FullName,
            id + ".txt");
        return File.ReadAllText(path);
    };
```

```
let workingDirectory = DirectoryInfo(Environment.CurrentDirectory)

let read id =
    let path =
        Path.Combine(workingDirectory.FullName, id.ToString() + ".txt")
    File.ReadAllText(path)
```

Dependency Inversion Principle


High-level modules should not depend on low-level modules. Both should depend on abstractions
Abstractions should not depend upon details. Details should depend upon abstractions

Favour Composition over inheritance


Composite

```
public void Save(int id, string message)
{
    new LogSavingStoreWriter().Save(id, message);
    this.Store.Save(id, message);
    this.Cache.Save(id, message);
    new LogSavedStoreWriter().Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id));
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}
```

```
public interface IStoreWriter
{
    void Save(int id, string message);
}
```

```
public class CompositeStoreWriter : IStoreWriter
{
    private readonly IStoreWriter[] writers;

    public CompositeStoreWriter(params IStoreWriter[] writers)
    {
        this.writers = writers;
    }

    public void Save(int id, string message)
    {
        foreach (var w in this.writers)
            w.Save(id, message);
    }
}
```

```
public void Save(int id, string message)
{
    new LogSavingStoreWriter().Save(id, message);
    this.Store.Save(id, message);
    this.Cache.Save(id, message);
    new LogSavedStoreWriter().Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id));
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}
```

```
public void Save(int id, string message)
{




}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id));
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}
```

```
public void Save(int id, string message)
{



}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id));
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}

protected virtual IStore Store
```

```
public void Save(int id, string message)
{


}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id));
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}

protected virtual IStore Store
{
```

```
public void Save(int id, string message)
{

}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id));
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}

protected virtual IStore Store
{
    get { return this.store; }
```

```
public void Save(int id, string message)
{
    this.Writer.Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id));
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}

protected virtual IStore Store
{
    get { return this.store; }
```


Decorator

```
public void Save(int id, string message)
{
    this.Writer.Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id));
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}

protected virtual IStore Store
{
    get { return this.store; }
```

```
public class StoreCache : IStoreCache, IStoreWriter
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;

    public StoreCache()
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
    }

    public virtual void Save(int id, string message)
    {
        var m = new Maybe<string>(message);
        this.cache.AddOrUpdate(id, m, (i, s) => m);
    }

    public virtual Maybe<string> GetOrAdd(
        int id, Func<int, Maybe<string>> messageFactory)
    {
        return this.cache.GetOrAdd(id, messageFactory);
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;

    public StoreCache(IStoreWriter writer)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
    }

    public virtual void Save(int id, string message)
    {
        var m = new Maybe<string>(message);
        this.cache.AddOrUpdate(id, m, (i, s) => m);
    }

    public virtual Maybe<string> GetOrAdd(
        int id, Func<int, Maybe<string>> messageFactory)
    {
        return this.cache.GetOrAdd(id, messageFactory);
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;


    public StoreCache(IStoreWriter writer)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
    }

    public virtual void Save(int id, string message)
    {
        var m = new Maybe<string>(message);
        this.cache.AddOrUpdate(id, m, (i, s) => m);
    }

    public virtual Maybe<string> GetOrAdd(
        int id, Func<int, Maybe<string>> messageFactory)
    {
        return this.cache.GetOrAdd(id, messageFactory);
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;

    public StoreCache(IStoreWriter writer)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
    }

    public virtual void Save(int id, string message)
    {
        var m = new Maybe<string>(message);
        this.cache.AddOrUpdate(id, m, (i, s) => m);
    }

    public virtual Maybe<string> GetOrAdd(
        int id, Func<int, Maybe<string>> messageFactory)
    {
        return this.cache.GetOrAdd(id, messageFactory);
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;

    public StoreCache(IStoreWriter writer)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();

    }

    public virtual void Save(int id, string message)
    {
        var m = new Maybe<string>(message);
        this.cache.AddOrUpdate(id, m, (i, s) => m);
    }

    public virtual Maybe<string> GetOrAdd(
        int id, Func<int, Maybe<string>> messageFactory)
    {
        return this.cache.GetOrAdd(id, messageFactory);
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;

    public StoreCache(IStoreWriter writer)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
    }

    public virtual void Save(int id, string message)
    {
        var m = new Maybe<string>(message);
        this.cache.AddOrUpdate(id, m, (i, s) => m);
    }

    public virtual Maybe<string> GetOrAdd(
        int id, Func<int, Maybe<string>> messageFactory)
    {
        return this.cache.GetOrAdd(id, messageFactory);
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;

    public StoreCache(IStoreWriter writer)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
    }

    public virtual void Save(int id, string message)
    {

        var m = new Maybe<string>(message);
        this.cache.AddOrUpdate(id, m, (i, s) => m);
    }

    public virtual Maybe<string> GetOrAdd(
        int id, Func<int, Maybe<string>> messageFactory)
    {
        return this.cache.GetOrAdd(id, messageFactory);
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;

    public StoreCache(IStoreWriter writer)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
    }

    public virtual void Save(int id, string message)
    {
        this.writer.Save(id, message);
        var m = new Maybe<string>(message);
        this.cache.AddOrUpdate(id, m, (i, s) => m);
    }

    public virtual Maybe<string> GetOrAdd(
        int id, Func<int, Maybe<string>> messageFactory)
    {
        return this.cache.GetOrAdd(id, messageFactory);
    }
}
```

```
public void Save(int id, string message)
{
    this.Writer.Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id));
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}

protected virtual IStore Store
{
    get { return this.store; }
```

```
public class StoreLogger : IStoreLogger
{
    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }

    public virtual void DidNotFind(int id)
    {
        Log.Debug("No message {id} found.", id);
    }

    public virtual void Returning(int id)
    {
        Log.Debug("Returning message {id}.", id);
```

```
public class StoreLogger : IStoreLogger
{

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }

    public virtual void DidNotFind(int id)
    {
        Log.Debug("No message {id} found.", id);
    }

    public virtual void Returning(int id)
    {
```

```
public class StoreLogger : IStoreLogger
{


    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }

    public virtual void DidNotFind(int id)
    {
        Log.Debug("No message {id} found.", id);
    }

    public virtual void Returning(int id)
```

```
public class StoreLogger : IStoreLogger
{



    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }

    public virtual void DidNotFind(int id)
    {
        Log.Debug("No message {id} found.", id);
    }
```

```
public class StoreLogger : IStoreLogger
{



    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }

    public virtual void DidNotFind(int id)
    {
        Log.Debug("No message {id} found.", id);
    }
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }

    public virtual void DidNotFind(int id)
    {
        Log.Debug("No message {id} found.", id);
    }
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{

    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }

    public virtual void DidNotFind(int id)
    {
        Log.Debug("No message {id} found.", id);
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{


    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }

    public virtual void DidNotFind(int id)
    {
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{



    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }

    public virtual void DidNotFind(int id)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{



    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    public StoreLogger(IStoreWriter writer)
    {
    }

    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{

    public StoreLogger(IStoreWriter writer)
    {
    }

    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
    }
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{


    public StoreLogger(IStoreWriter writer)
    {
    }

    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;

    public StoreLogger(IStoreWriter writer)
    {
    }

    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
        Log.Debug("Reading message {id}.", id);
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;

    public StoreLogger(IStoreWriter writer)
    {

    }

    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;

    public StoreLogger(IStoreWriter writer)
    {
        this.writer = writer;
    }

    public void Save(int id, string message)
    {
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
    {
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;

    public StoreLogger(IStoreWriter writer)
    {
        this.writer = writer;
    }

    public void Save(int id, string message)
    {

    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;

    public StoreLogger(IStoreWriter writer)
    {
        this.writer = writer;
    }

    public void Save(int id, string message)
    {
        this.writer.Save(id, message);
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Reading(int id)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;

    public StoreLogger(IStoreWriter writer)
    {
        this.writer = writer;
    }

    public void Save(int id, string message)
    {

        this.writer.Save(id, message);

    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;

    public StoreLogger(IStoreWriter writer)
    {
        this.writer = writer;
    }

    public void Save(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
        this.writer.Save(id, message);
        Log.Information("Saved message {id}.", id);
    }

    public virtual void Saving(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
    }

    public virtual void Saved(int id, string message)
    {
        Log.Information("Saved message {id}.", id);
    }
```

```
public MessageStore(DirectoryInfo workingDirectory)
{
    this.WorkingDirectory = workingDirectory;
    var fileStore = new FileStore(workingDirectory);
    var c = new StoreCache(fileStore);
    this.cache = c;
    var l = new StoreLogger(c);
    this.log = l;
    this.store = fileStore;
    this.fileLocator = fileStore;
    this.writer = new CompositeStoreWriter(l);
}

public DirectoryInfo WorkingDirectory { get; private set; }

public void Save(int id, string message)
{
    this.Writer.Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id    => this.Store.ReadAllText(id));
```

```
public MessageStore(DirectoryInfo workingDirectory)
{
    this.WorkingDirectory = workingDirectory;
    var fileStore = new FileStore(workingDirectory);
    var c = new StoreCache(fileStore);
    this.cache = c;
    var l = new StoreLogger(c);
    this.log = l;
    this.store = fileStore;
    this.fileLocator = fileStore;
    this.writer = new CompositeStoreWriter(l);
}

public DirectoryInfo WorkingDirectory { get; private set; }

public void Save(int id, string message)
{
    this.Writer.Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id    => this.Store.ReadAllText(id));
```

```
public MessageStore(DirectoryInfo workingDirectory)
{
    this.WorkingDirectory = workingDirectory;
    var fileStore = new FileStore(workingDirectory);
    var c = new StoreCache(fileStore);
    this.cache = c;
    var l = new StoreLogger(c);
    this.log = l;
    this.store = fileStore;
    this.fileLocator = fileStore;
    this.writer = l;
}

public DirectoryInfo WorkingDirectory { get; private set; }

public void Save(int id, string message)
{
    this.Writer.Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(
        id    => this.Store.ReadAllText(id));
```

```
public interface IStoreCache
{
    void Save(int id, string message);

    Maybe<string> GetOrAdd(int id,
        Func<int, Maybe<string>> messageFactory);
}
```

```
public interface IStoreCache
{
    void Save(int id, string message);

    Maybe<string> GetOrAdd(int id,
        Func<int, Maybe<string>> messageFactory);
}
```

```
public interface IStoreCache
{
    void Save(int id, string message);

    Maybe<string> GetOrAdd(int id,
                                          );
}
```

```
public interface IStoreCache
{
    void Save(int id, string message);

    Maybe<string> GetOrAdd(int id);
}
```

```
public void Save(int id, string message)
{
    this.Writer.Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.GetOrAdd(id);
    if (!message.Any())
    {
        message = this.Store.ReadAllText(id);
        if (message.Any())
            this.Cache.Save(id, message.Single());
    }
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}
```

```
public interface IStoreCache
{
    void Save(int id, string message);

    Maybe<string> GetOrAdd(int id);
}
```

```
public interface IStoreCache
{
    void Save(int id, string message);

    Maybe<string> GetOrAdd(int id);
}
```

```
public interface IStoreCache
{
    void Save(int id, string message);

    Maybe<string> Read(int id);
}
```

```
public interface IStore
{
    void Save(int id, string message);

    Maybe<string> ReadAllText(int id);
}
```

```
public interface IStore
{
    void Save(int id, string message);

    Maybe<string> ReadAllText(int id);
}
```

```
public interface IStore
{
    void Save(int id, string message);

    Maybe<string> Read(int id);
}
```

```
public void Save(int id, string message)
{
    this.Writer.Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.Read(id);
    if (!message.Any())
    {
        message = this.Store.Read(id);
        if (message.Any())
            this.Cache.Save(id, message.Single());
    }
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}
```

```
public interface IStore
{
    void Save(int id, string message);

    Maybe<string> Read(int id);
}
```

```
public interface IStore
{
    void Save(int id, string message);

    Maybe<string> Read(int id);
}
```

```
public interface IStoreReader
{
    Maybe<string> Read(int id);
}
```

```
public class StoreCache : IStoreCache, IStoreWriter
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;

    public StoreCache(IStoreWriter writer)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;
        return new Maybe<string>();
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter, IStoreReader
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;

    public StoreCache(IStoreWriter writer)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;
        return new Maybe<string>();
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter, IStoreReader
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;

    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;
        return new Maybe<string>();
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter, IStoreReader
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;


    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;
        return new Maybe<string>();
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter, IStoreReader
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;
        return new Maybe<string>();
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter, IStoreReader
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;

    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;
        return new Maybe<string>();
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter, IStoreReader
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
        this.reader = reader;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;
        return new Maybe<string>();
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter, IStoreReader
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
        this.reader = reader;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;

        return new Maybe<string>();
    }
}
```

```
public class StoreCache : IStoreCache, IStoreWriter, IStoreReader
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
        this.reader = reader;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;


        return new Maybe<string>();
    }
}
```

```
{
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
        this.reader = reader;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;



        return new Maybe<string>();
    }
}
```

```
    private readonly ConcurrentDictionary<int, Maybe<string>> cache;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
        this.reader = reader;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;




        return new Maybe<string>();
    }
}
```

```
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
        this.reader = reader;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;




        return new Maybe<string>();
    }
}
```

```
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreCache(IStoreWriter writer, IStoreReader reader)
    {
        this.cache = new ConcurrentDictionary<int, Maybe<string>>();
        this.writer = writer;
        this.reader = reader;
    }

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
    {
        Maybe<string> retVal;
        if (this.cache.TryGetValue(id, out retVal))
            return retVal;

        retVal = this.reader.Read(id);
        if (retVal.Any())
            this.cache.AddOrUpdate(id, retVal, (i, s) => retVal);

        return retVal;
    }
}
```

```
public void Save(int id, string message)
{
    this.Writer.Save(id, message);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var message = this.Cache.Read(id);
    if (message.Any())
        this.Log.Returning(id);
    else
        this.Log.DidNotFind(id);
    return message;
}

public FileInfo GetFileInfo(int id)
{
    return this.FileLocator.GetFileInfo(id);
}

protected virtual IStore Store
{
    get { return this.store; }
}
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;

    public StoreLogger(IStoreWriter writer)
    {
        this.writer = writer;
    }

    public void Save(int id, string message)

    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)

    public virtual void Returning(int id)
}
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
    }

    public void Save(int id, string message)

    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)

    public virtual void Returning(int id)
}
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;


    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
    }

    public void Save(int id, string message)

    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)

    public virtual void Returning(int id)
}
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
    }

    public void Save(int id, string message)

    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)

    public virtual void Returning(int id)
}
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;

    }

    public void Save(int id, string message)

    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)

    public virtual void Returning(int id)
}
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)

    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)

    public virtual void Returning(int id)
}
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)


    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)

    public virtual void Returning(int id)
}
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)



    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)

    public virtual void Returning(int id)
}
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)



    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)

    public virtual void Returning(int id)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)




    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)




    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)

    public virtual void DidNotFind(int id)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)




    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter, IStoreReader
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)

    public Maybe<string> Read(int id)
    {
        var retVal = this.reader.Read(id);
        return retVal;
    }

    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter, IStoreReader
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)

    public Maybe<string> Read(int id)
    {
        var retVal = this.reader.Read(id);

        return retVal;
    }

    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)

    public virtual void Reading(int id)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter, IStoreReader
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)

    public Maybe<string> Read(int id)
    {
        var retVal = this.reader.Read(id);


        return retVal;
    }

    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter, IStoreReader
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)

    public Maybe<string> Read(int id)
    {
        var retVal = this.reader.Read(id);



        return retVal;
    }

    public virtual void Saving(int id, string message)

    public virtual void Saved(int id, string message)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter, IStoreReader
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)

    public Maybe<string> Read(int id)
    {

        var retVal = this.reader.Read(id);




        return retVal;
    }

    public virtual void Saving(int id, string message)
```

```
public class StoreLogger : IStoreLogger, IStoreWriter, IStoreReader
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)

    public Maybe<string> Read(int id)
    {
        Log.Debug("Reading message {id}.", id);
        var retVal = this.reader.Read(id);
        if (retVal.Any())
            Log.Debug("Returning message {id}.", id);
        else
            Log.Debug("No message {id} found.", id);
        return retVal;
    }

    public virtual void Saving(int id, string message)
```

```
public MessageStore(DirectoryInfo workingDirectory)
{
    this.WorkingDirectory = workingDirectory;
    var fileStore = new FileStore(workingDirectory);
    var c = new StoreCache(fileStore, fileStore);
    this.cache = c;
    var l = new StoreLogger(c, c);
    this.log = l;
    this.store = fileStore;
    this.fileLocator = fileStore;
    this.writer = l;
    this.reader = l;
}

public DirectoryInfo WorkingDirectory { get; private set; }

public void Save(int id, string message)
{
    this.Writer.Save(id, message);
}

public Maybe<string> Read(int id)
{
    return this.Reader.Read(id);
}
```


Final clean-up

```
public class MessageStore
{
    private readonly IStoreCache cache;
    private readonly IStoreLogger log;
    private readonly IStore store;
    private readonly IFileLocator fileLocator;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public MessageStore(DirectoryInfo workingDirectory)
    {
        this.WorkingDirectory = workingDirectory;
        var fileStore = new FileStore(workingDirectory);
        var c = new StoreCache(fileStore, fileStore);
        this.cache = c;
        var l = new StoreLogger(c, c);
        this.log = l;
        this.store = fileStore;
        this.fileLocator = fileStore;
        this.writer = l;
        this.reader = l;
    }

    public DirectoryInfo WorkingDirectory { get; private set; }
```

```
public class MessageStore
{
    private readonly IFileLocator fileLocator;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public MessageStore(
        IStoreWriter writer,
        IStoreReader reader,
        IFileLocator fileLocator)
    {
        if (writer == null)
            throw new ArgumentNullException("writer");
        if (reader == null)
            throw new ArgumentNullException("reader");
        if (fileLocator == null)
            throw new ArgumentNullException("fileLocator");

        this.fileLocator = fileLocator;
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)
    {
```

```
public class StoreCache : IStoreWriter, IStoreReader
{
    public StoreCache(IStoreWriter writer, IStoreReader reader)

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
}
```

```
public class StoreCache : IStoreWriter, IStoreReader
{
    public StoreCache(IStoreWriter writer, IStoreReader reader)

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)
}
```

```
public class StoreCache : IStoreWriter, IStoreReader
{
    public StoreCache(IStoreWriter writer, IStoreReader reader)

    public void Save(int id, string message)

    public Maybe<string> Read(int id)
}
```

```
public class FileStore : IFileLocator, IStoreWriter, IStoreReader
{
    public FileStore(DirectoryInfo workingDirectory)

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)

    public virtual FileInfo GetFileInfo(int id)
}
```

```
public class FileStore : IFileLocator, IStoreWriter, IStoreReader
{
    public FileStore(DirectoryInfo workingDirectory)

    public virtual void Save(int id, string message)

    public virtual Maybe<string> Read(int id)

    public virtual FileInfo GetFileInfo(int id)
}
```

```
public class FileStore : IFileLocator, IStoreWriter, IStoreReader
{
    public FileStore(DirectoryInfo workingDirectory)

    public void Save(int id, string message)

    public Maybe<string> Read(int id)

    public FileInfo GetFileInfo(int id)
}
```

```
public class StoreLogger : IStoreWriter, IStoreReader
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(
        IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
        this.writer.Save(id, message);
        Log.Information("Saved message {id}.", id);
    }

    public Maybe<string> Read(int id)
    {
        Log.Debug("Reading message {id}.", id);
        var retVal = this.reader.Read(id);
        if (retVal.Any())
            Log.Debug("Returning message {id}.", id);
```

```
public class StoreLogger : IStoreWriter, IStoreReader
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(
                    IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
        this.writer.Save(id, message);
        Log.Information("Saved message {id}.", id);
    }

    public Maybe<string> Read(int id)
    {
        Log.Debug("Reading message {id}.", id);
        var retVal = this.reader.Read(id);
        if (retVal.Any())
            Log.Debug("Returning message {id}.", id);
```

```
public class StoreLogger : IStoreWriter, IStoreReader
{
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(
        ILogger log, IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
        this.writer.Save(id, message);
        Log.Information("Saved message {id}.", id);
    }

    public Maybe<string> Read(int id)
    {
        Log.Debug("Reading message {id}.", id);
        var retVal = this.reader.Read(id);
        if (retVal.Any())
            Log.Debug("Returning message {id}.", id);
```

```
public class StoreLogger : IStoreWriter, IStoreReader
{

    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(
        ILogger log, IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
        this.writer.Save(id, message);
        Log.Information("Saved message {id}.", id);
    }

    public Maybe<string> Read(int id)
    {
        Log.Debug("Reading message {id}.", id);
        var retVal = this.reader.Read(id);
        if (retVal.Any())
```

```
public class StoreLogger : IStoreWriter, IStoreReader
{
    private readonly ILogger log;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(
        ILogger log, IStoreWriter writer, IStoreReader reader)
    {
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
        this.writer.Save(id, message);
        Log.Information("Saved message {id}.", id);
    }

    public Maybe<string> Read(int id)
    {
        Log.Debug("Reading message {id}.", id);
        var retVal = this.reader.Read(id);
        if (retVal.Any())
```

```
public class StoreLogger : IStoreWriter, IStoreReader
{
    private readonly ILogger log;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(
        ILogger log, IStoreWriter writer, IStoreReader reader)
    {

        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
        this.writer.Save(id, message);
        Log.Information("Saved message {id}.", id);
    }

    public Maybe<string> Read(int id)
    {
        Log.Debug("Reading message {id}.", id);
        var retVal = this.reader.Read(id);
```

```
public class StoreLogger : IStoreWriter, IStoreReader
{
    private readonly ILogger log;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(
        ILogger log, IStoreWriter writer, IStoreReader reader)
    {
        this.log = log;
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
        this.writer.Save(id, message);
        Log.Information("Saved message {id}.", id);
    }

    public Maybe<string> Read(int id)
    {
        Log.Debug("Reading message {id}.", id);
        var retVal = this.reader.Read(id);
```

```
public class StoreLogger : IStoreWriter, IStoreReader
{
    private readonly ILogger log;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(
        ILogger log, IStoreWriter writer, IStoreReader reader)
    {
        this.log = log;
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)
    {
        Log.Information("Saving message {id}.", id);
        this.writer.Save(id, message);
        Log.Information("Saved message {id}.", id);
    }

    public Maybe<string> Read(int id)
    {
        Log.Debug("Reading message {id}.", id);
        var retVal = this.reader.Read(id);
```

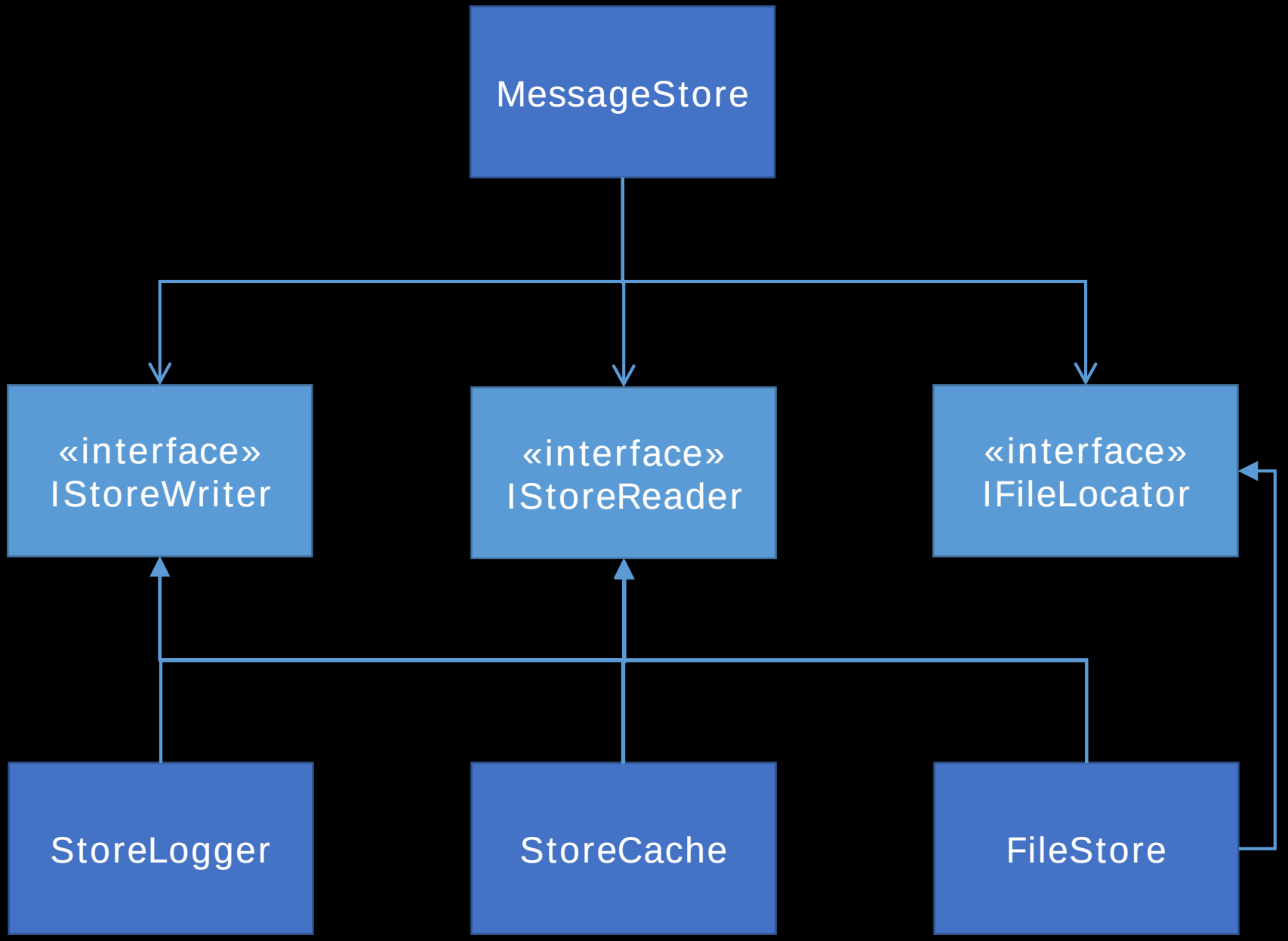
```
public class StoreLogger : IStoreWriter, IStoreReader
{
    private readonly ILogger log;
    private readonly IStoreWriter writer;
    private readonly IStoreReader reader;

    public StoreLogger(
        ILogger log, IStoreWriter writer, IStoreReader reader)
    {
        this.log = log;
        this.writer = writer;
        this.reader = reader;
    }

    public void Save(int id, string message)
    {
        this.log.Information("Saving message {id}.", id);
        this.writer.Save(id, message);
        this.log.Information("Saved message {id}.", id);
    }

    public Maybe<string> Read(int id)
    {
        this.log.Debug("Reading message {id}.", id);
        var retVal = this.reader.Read(id);
```

```
var logger = new LoggerConfiguration().CreateLogger();
var fileStore =
    new FileStore(
        new DirectoryInfo(
            Environment.CurrentDirectory));
var cache = new StoreCache(fileStore, fileStore);
var log = new StoreLogger(logger, cache, cache);
var msgStore = new MessageStore(
    log,
    log,
    fileStore);
```


MessageStore
«interface»
IStoreWriter
«interface»
IStoreReader
«interface»
IFileLocator
StoreLogger
StoreCache
FileStore

Conclusion

Single Responsibility Principle

Open Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle


Not a goal


A reaction to
design smells


Supple
rather than solid