Append-only

Create
Read
Update
Delete

Create

Read

Update

Delete

Create

Read

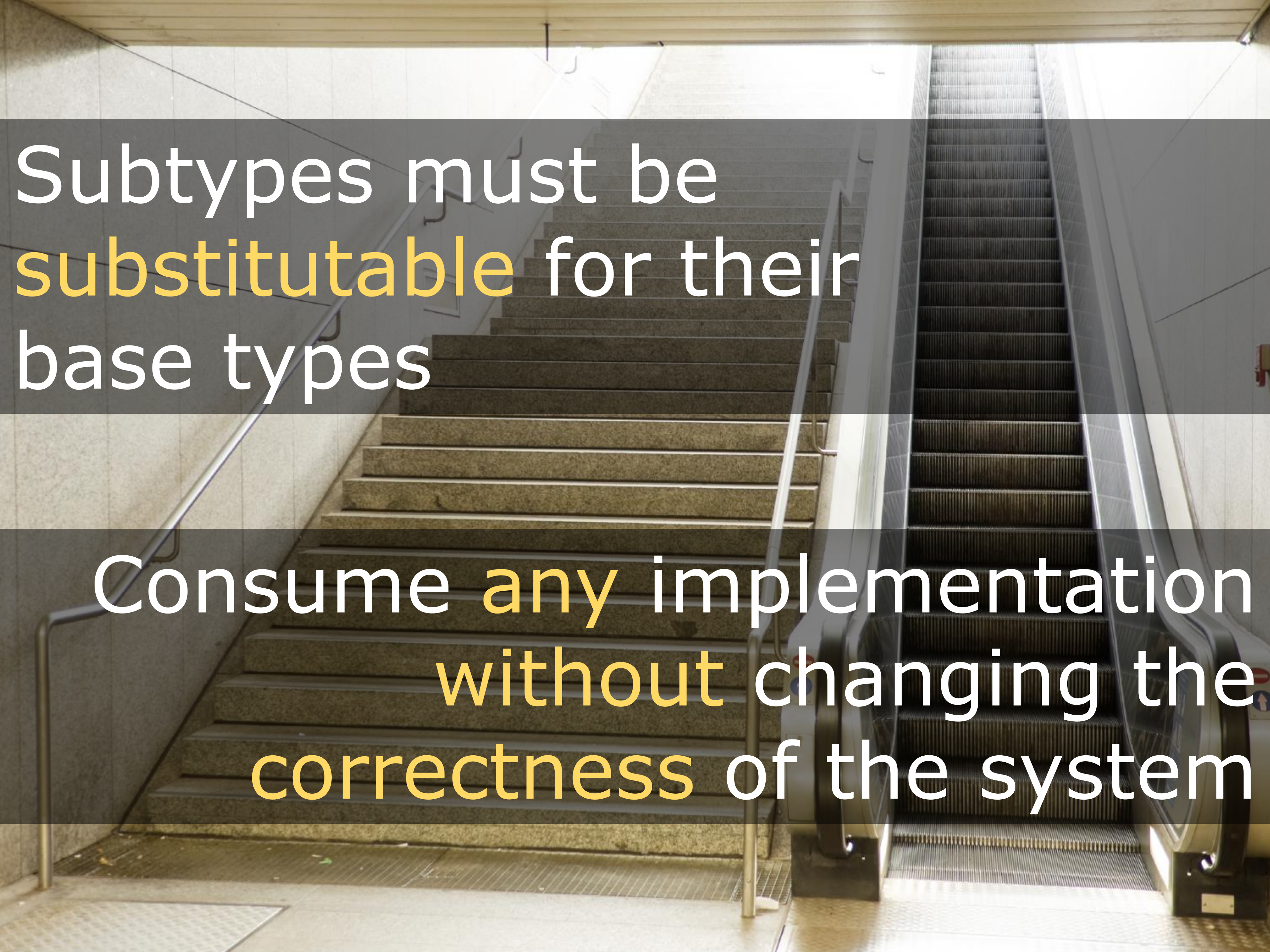
Create

Read

How do you evolve such a code base?

Strangler

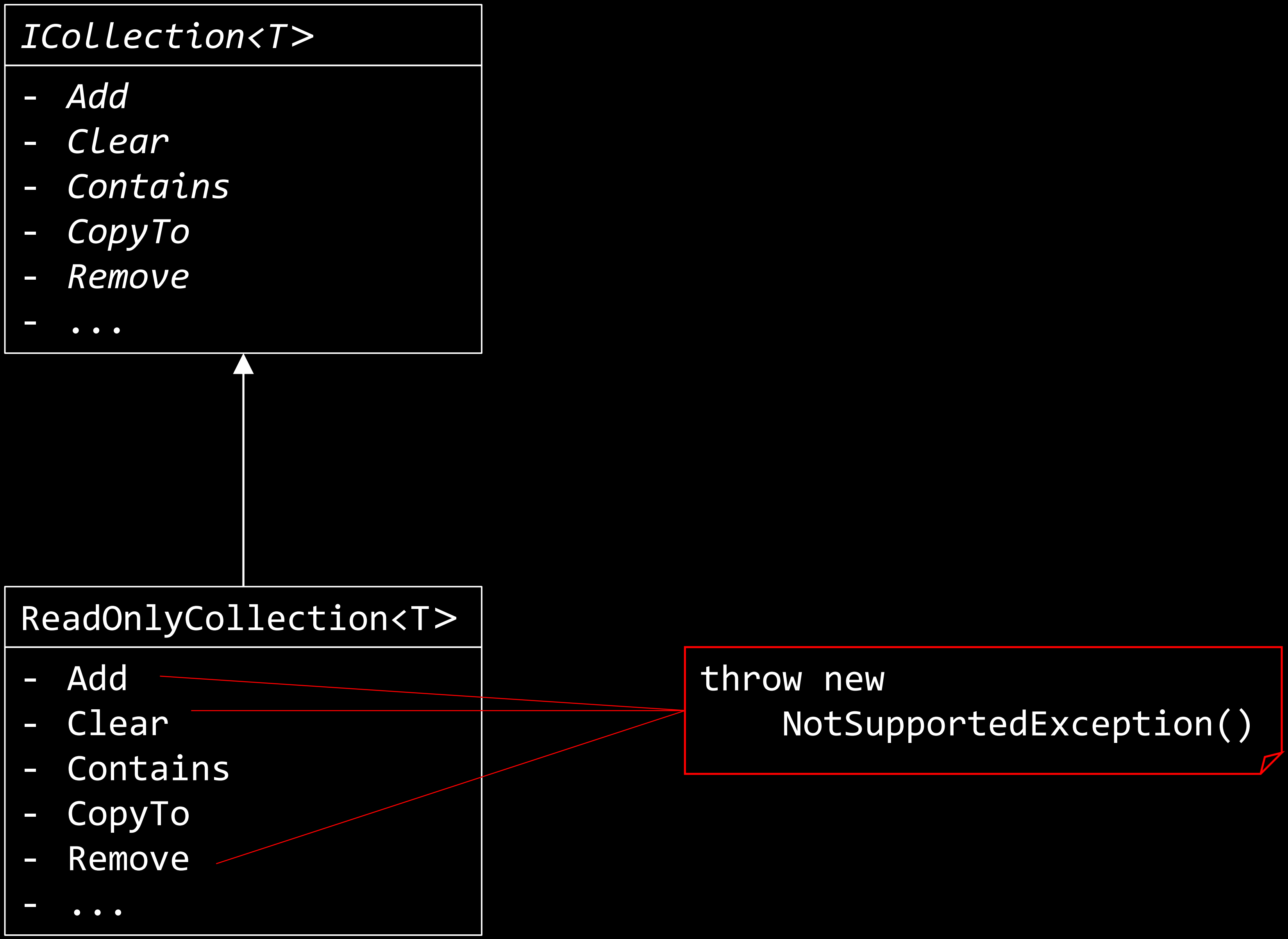
Liskov Substitution Principle



Subtypes must be substitutable for their base types

Consume any implementation without changing the correctness of the system

NotSupportedException

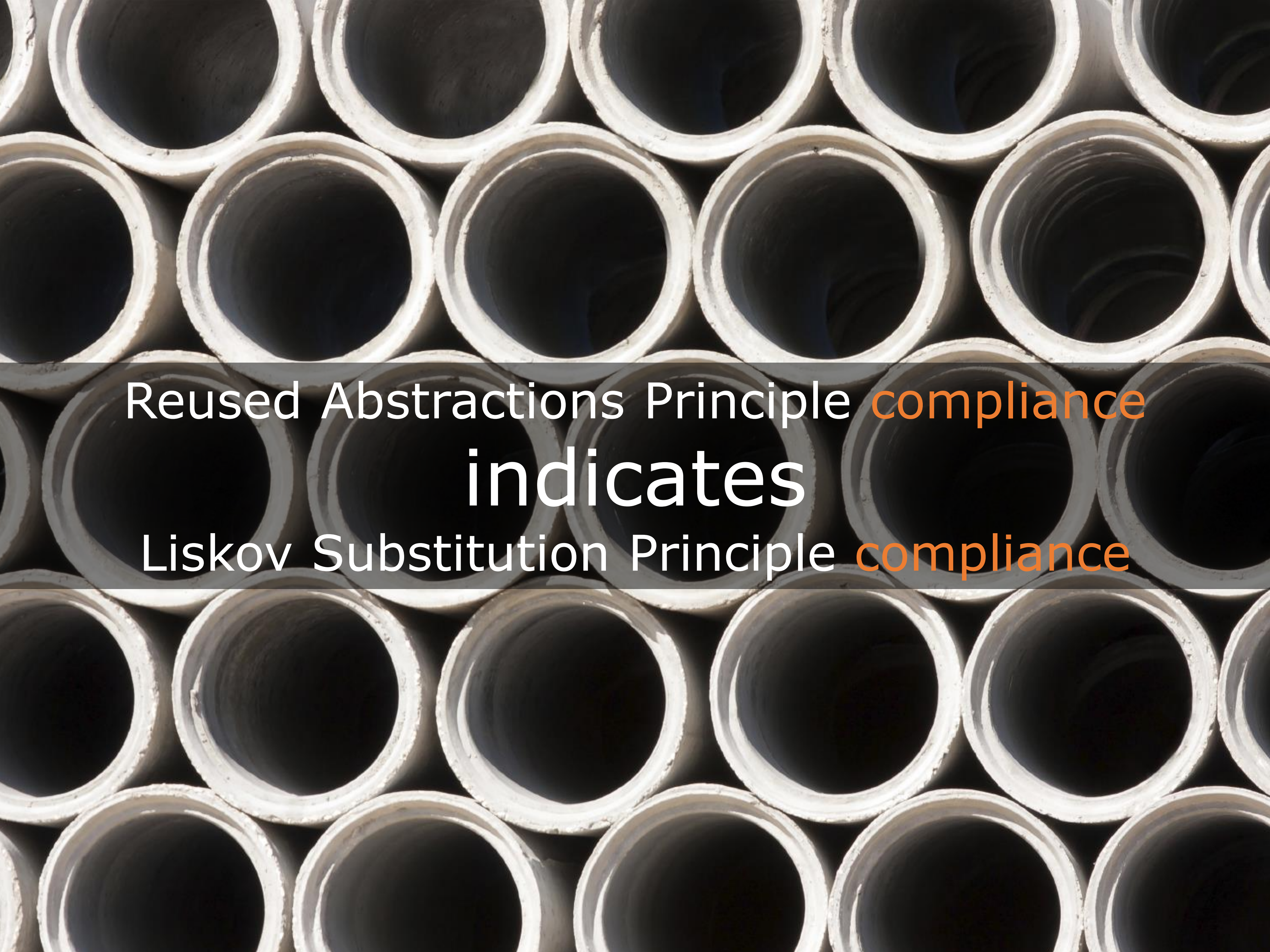

ICollection<T>
- *Add*
- *Clear*
- *Contains*
- *CopyTo*
- *Remove*
- ...
ReadOnlyCollection<T>
- Add
- Clear
- Contains
- CopyTo
- Remove
- ...
throw new NotSupportedException()

NotSupportedException

Downcasts

Extracted interfaces

LSP is often violated by attempts to remove features



Reused Abstractions Principle compliance

indicates

Liskov Substitution Principle compliance

```
public void Save(int id, string message)
{
    this.Log.Saving(id);
    var file = this.GetFileInfo(id);
    this.Store.WriteAllText(file.FullName, message);
    this.Cache.AddOrUpdate(id, message);
    this.Log.Saved(id);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var file = this.GetFileInfo(id);
    if (!file.Exists)
    {
        this.Log.DidNotFind(id);
        return new Maybe<string>();
    }
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(file.FullName));
    this.Log.Returning(id);
    return new Maybe<string>(message);
}
```

```csharp
public class FileStore
{
    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```
public class SqlStore : FileStore
{
    public override void WriteAllText(string path, string message)
    {
        // Write to database here
    }

    public override string ReadAllText(string path)
    {
        // Read and return from database here
    }

    public override FileInfo GetFileInfo(int id, string workingDirectory
    {
        return base.GetFileInfo(id, workingDirectory);
    }
}
```

```
public class FileStore
{
    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```csharp
public class FileStore
{
    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```
public interface IStore
{
    void WriteAllText(string path, string message);

    string ReadAllText(string path);

    FileInfo GetFileInfo(int id, string workingDirectory);
}
```

```
public class MessageStore
{
    private readonly StoreCache cache;
    private readonly StoreLogger log;
    private readonly FileStore fileStore;

    public MessageStore(DirectoryInfo workingDirectory)
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");
        if (!workingDirectory.Exists)
            throw new ArgumentException("Boo", "workingDirectory");

        this.WorkingDirectory = workingDirectory;
        this.cache = new StoreCache();
        this.log = new StoreLogger();
        this.fileStore = new FileStore();
    }

    public DirectoryInfo WorkingDirectory { get; private set; }

    public void Save(int id, string message)
    {
        this.log.Saving(id);
        var file = this.GetFileInfo(id);
```

```
public class MessageStore
{
    private readonly StoreCache cache;
    private readonly StoreLogger log;
    private readonly FileStore fileStore;

    public MessageStore(DirectoryInfo workingDirectory)
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");
        if (!workingDirectory.Exists)
            throw new ArgumentException("Boo", "workingDirectory");

        this.WorkingDirectory = workingDirectory;
        this.cache = new StoreCache();
        this.log = new StoreLogger();
        this.fileStore = new FileStore();
    }

    public DirectoryInfo WorkingDirectory { get; private set; }

    public void Save(int id, string message)
    {
        this.log.Saving(id);
        var file = this.GetFileInfo(id);
```

```
public class MessageStore
{
    private readonly StoreCache cache;
    private readonly StoreLogger log;
    private readonly IStore store;

    public MessageStore(DirectoryInfo workingDirectory)
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");
        if (!workingDirectory.Exists)
            throw new ArgumentException("Boo", "workingDirectory");

        this.WorkingDirectory = workingDirectory;
        this.cache = new StoreCache();
        this.log = new StoreLogger();
        this.store = new FileStore();
    }

    public DirectoryInfo WorkingDirectory { get; private set; }

    public void Save(int id, string message)
    {
        this.log.Saving(id);
        var file = this.GetFileInfo(id);
```

```
public class SqlStore : IStore
{
    public void WriteAllText(string path, string message)
    {
        // Write to database here
    }

    public string ReadAllText(string path)
    {
        // Read and return from database here
    }

    public FileInfo GetFileInfo(int id, string workingDirectory)
    {
        throw new NotSupportedException();
    }
}
```

```csharp
public void Save(int id, string message)
{
    this.log.Saving(id);
    var file = this.GetFileInfo(id);
    this.fileStore.WriteAllText(file.FullName, message);
    this.cache.AddOrUpdate(id, message);
    this.log.Saved(id);
}

public Maybe<string> Read(int id)
{
    this.log.Reading(id);
    var file = this.GetFileInfo(id);
    if (!file.Exists)
    {
        this.log.DidNotFind(id);
        return new Maybe<string>();
    }
    var message = this.cache.GetOrAdd(
        id, _ => this.fileStore.ReadAllText(file.FullName));
    this.log.Returning(id);
    return new Maybe<string>(message);
}
```

```csharp
public void Save(int id, string message)
{
    this.log.Saving(id);
    var file = this.GetFileInfo(id);
    this.fileStore.WriteAllText(file.FullName, message);
    this.cache.AddOrUpdate(id, message);
    this.log.Saved(id);
}

public Maybe<string> Read(int id)
{
    this.log.Reading(id);
    var file = this.GetFileInfo(id);
    if (!file.Exists)
    {
        this.log.DidNotFind(id);
        return new Maybe<string>();
    }
    var message = this.cache.GetOrAdd(
        id, _ => this.fileStore.ReadAllText(file.FullName));
    this.log.Returning(id);
    return new Maybe<string>(message);
}
```

```
public class SqlStore : IStore
{
    public void WriteAllText(string path, string message)
    {
        // Write to database here
    }

    public string ReadAllText(string path)
    {
        // Read and return from database here
    }

    public FileInfo GetFileInfo(int id, string workingDirectory)
    {
        throw new NotSupportedException();
    }
}
```

```csharp
public class SqlStore : IStore
{
    public void WriteAllText(string path, string message)
    {
        // Write to database here
    }

    public string ReadAllText(string path)
    {
        // Read and return from database here
    }

    public FileInfo GetFileInfo(int id, string workingDirectory)
    {
        // Return a bogus FileInfo here
    }
}
```

```
public void Save(int id, string message)
{
    this.log.Saving(id);
    var file = this.GetFileInfo(id);
    this.fileStore.WriteAllText(file.FullName, message);
    this.cache.AddOrUpdate(id, message);
    this.log.Saved(id);
}

public Maybe<string> Read(int id)
{
    this.log.Reading(id);
    var file = this.GetFileInfo(id);
    if (!file.Exists)
    {
        this.log.DidNotFind(id);
        return new Maybe<string>();
    }
    var message = this.cache.GetOrAdd(
        id, _ => this.fileStore.ReadAllText(file.FullName));
    this.log.Returning(id);
    return new Maybe<string>(message);
}
```

```
public void Save(int id, string message)
{
    this.log.Saving(id);
    var file = this.GetFileInfo(id);
    this.fileStore.WriteAllText(file.FullName, message);
    this.cache.AddOrUpdate(id, message);
    this.log.Saved(id);
}

public Maybe<string> Read(int id)
{
    this.log.Reading(id);
    var file = this.GetFileInfo(id);
    if (!file.Exists)
    {
        this.log.DidNotFind(id);
        return new Maybe<string>();
    }
    var message = this.cache.GetOrAdd(
        id, _ => this.fileStore.ReadAllText(file.FullName));
    this.log.Returning(id);
    return new Maybe<string>(message);
}
```

```csharp
public class FileStore : IStore
{
    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```csharp
public class FileStore : IStore
{

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```csharp
public class FileStore : IStore
{


    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```csharp
public class FileStore : IStore
{



    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```csharp
public class FileStore : IStore
{




    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```
public class FileStore : IStore
{
    public FileStore()
    {
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```
public class FileStore : IStore
{
    public FileStore(DirectoryInfo workingDirectory)
    {
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```
public class FileStore : IStore
{

    public FileStore(DirectoryInfo workingDirectory)
    {
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```
public class FileStore : IStore
{


    public FileStore(DirectoryInfo workingDirectory)
    {
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

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

    public FileStore(DirectoryInfo workingDirectory)
    {
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

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

    public FileStore(DirectoryInfo workingDirectory)
    {

    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

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

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

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

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

    public FileStore(DirectoryInfo workingDirectory)
    {

        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
```

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

    public FileStore(DirectoryInfo workingDirectory)
    {


        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
```

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

    public FileStore(DirectoryInfo workingDirectory)
    {



        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
public class FileStore : IStore
{
    private readonly DirectoryInfo workingDirectory;

    public FileStore(DirectoryInfo workingDirectory)
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");

        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
public class FileStore : IStore
{
    private readonly DirectoryInfo workingDirectory;

    public FileStore(DirectoryInfo workingDirectory)
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");


        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
```

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

    public FileStore(DirectoryInfo workingDirectory)
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");



        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
public class FileStore : IStore
{
    private readonly DirectoryInfo workingDirectory;

    public FileStore(DirectoryInfo workingDirectory)
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");
        if (!workingDirectory.Exists)
            throw new ArgumentException("Boo", "workingDirectory");

        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");
        if (!workingDirectory.Exists)
            throw new ArgumentException("Boo", "workingDirectory");

        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(workingDirectory, id + ".txt"));
    }
}
```

```
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");
        if (!workingDirectory.Exists)
            throw new ArgumentException("Boo", "workingDirectory");

        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(this.workingDirectory.FullName, id + ".txt"));
    }
}
```

```
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");
        if (!workingDirectory.Exists)
            throw new ArgumentException("Boo", "workingDirectory");

        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id, string workingDirectory)
    {
        return new FileInfo(
            Path.Combine(this.workingDirectory.FullName, id + ".txt"));
    }
}
```

```
    {
        if (workingDirectory == null)
            throw new ArgumentNullException("workingDirectory");
        if (!workingDirectory.Exists)
            throw new ArgumentException("Boo", "workingDirectory");

        this.workingDirectory = workingDirectory;
    }

    public virtual void WriteAllText(string path, string message)
    {
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)
    {
        return File.ReadAllText(path);
    }

    public virtual FileInfo GetFileInfo(int id)
    {
        return new FileInfo(
            Path.Combine(this.workingDirectory.FullName, id + ".txt"));
    }
}
```

```
public interface IStore
{
    void WriteAllText(string path, string message);

    string ReadAllText(string path);

    FileInfo GetFileInfo(int id);
}
```

```
public interface IStore
{
    void WriteAllText(string path, string message);

    string ReadAllText(string path);

    FileInfo GetFileInfo(int id);
}
```

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

    string ReadAllText(string path);

    FileInfo GetFileInfo(int id);
}
```

```
public class FileStore : IStore
{
    public FileStore(DirectoryInfo workingDirectory)

    public virtual void WriteAllText(int id, string message)
    {
        var path = this.GetFileInfo(id).FullName;
        File.WriteAllText(path, message);
    }

    public virtual string ReadAllText(string path)

    public virtual FileInfo GetFileInfo(int id)
    {
        return new FileInfo(
            Path.Combine(this.workingDirectory.FullName, id + ".txt"));
    }
}
```

```
public class MessageStore
{
    public MessageStore(DirectoryInfo workingDirectory)

    public DirectoryInfo WorkingDirectory { get; }

    public void Save(int id, string message)
    {
        this.Log.Saving(id);
        var file = this.GetFileInfo(id);
        this.Store.WriteAllText(id, message);
        this.Cache.AddOrUpdate(id, message);
        this.Log.Saved(id);
    }

    public Maybe<string> Read(int id)

    public FileInfo GetFileInfo(int id)

    protected virtual IStore Store { get; }

    protected virtual StoreCache Cache { get; }

    protected virtual StoreLogger Log { get; }
}
```

```csharp
public class MessageStore
{
    public MessageStore(DirectoryInfo workingDirectory)

    public DirectoryInfo WorkingDirectory { get; }

    public void Save(int id, string message)
    {
        this.Log.Saving(id);
        var file = this.GetFileInfo(id);
        this.Store.WriteAllText(id, message);
        this.Cache.AddOrUpdate(id, message);
        this.Log.Saved(id);
    }

    public Maybe<string> Read(int id)

    public FileInfo GetFileInfo(int id)

    protected virtual IStore Store { get; }

    protected virtual StoreCache Cache { get; }

    protected virtual StoreLogger Log { get; }
}
```

```csharp
public class MessageStore
{
    public MessageStore(DirectoryInfo workingDirectory)

    public DirectoryInfo WorkingDirectory { get; }

    public void Save(int id, string message)
    {
        this.Log.Saving(id);

        this.Store.WriteAllText(id, message);
        this.Cache.AddOrUpdate(id, message);
        this.Log.Saved(id);
    }

    public Maybe<string> Read(int id)

    public FileInfo GetFileInfo(int id)

    protected virtual IStore Store { get; }

    protected virtual StoreCache Cache { get; }

    protected virtual StoreLogger Log { get; }
}
```

```
public class MessageStore
{
    public MessageStore(DirectoryInfo workingDirectory)

    public DirectoryInfo WorkingDirectory { get; }

    public void Save(int id, string message)
    {
        this.Log.Saving(id);
        this.Store.WriteAllText(id, message);
        this.Cache.AddOrUpdate(id, message);
        this.Log.Saved(id);
    }

    public Maybe<string> Read(int id)

    public FileInfo GetFileInfo(int id)

    protected virtual IStore Store { get; }

    protected virtual StoreCache Cache { get; }

    protected virtual StoreLogger Log { get; }
}
```

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

    string ReadAllText(string path);

    FileInfo GetFileInfo(int id);
}
```

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

    string ReadAllText(string path);

    FileInfo GetFileInfo(int id);
}
```

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

    string ReadAllText(int id);

    FileInfo GetFileInfo(int id);
}
```

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

    string ReadAllText(int id);

    FileInfo GetFileInfo(int id);
}
```

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

    Maybe<string> ReadAllText(int id);

    FileInfo GetFileInfo(int id);
}
```

```
    this.Cache.AddOrUpdate(id, message);
    this.Log.Saved(id);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var file = this.GetFileInfo(id);
    if (!file.Exists)
    {
        this.Log.DidNotFind(id);
        return new Maybe<string>();
    }
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id).Single());
    this.Log.Returning(id);
    return new Maybe<string>(message);
}

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

protected virtual IStore Store
{
    this.Cache.AddOrUpdate(id, message);
    this.Log.Saved(id);
}

public Maybe<string> Read(int id)
{
    this.Log.Reading(id);
    var file = this.GetFileInfo(id);
    if (!file.Exists)
    {
        this.Log.DidNotFind(id);
        return new Maybe<string>();
    }
    var message = this.Cache.GetOrAdd(
        id, _ => this.Store.ReadAllText(id).Single());
    this.Log.Returning(id);
    return new Maybe<string>(message);
}

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

protected virtual IStore Store
{
```

```
public class StoreCache
{
    private readonly ConcurrentDictionary<int, string> cache;

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

    public virtual void AddOrUpdate(int id, string message)
    {
        this.cache.AddOrUpdate(id, message, (i, s) => message);
    }

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

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

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

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

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

    public virtual void AddOrUpdate(int id, string message)
    {
        this.cache.AddOrUpdate(id, message, (i, s) => message);
    }

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

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

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

    public virtual void AddOrUpdate(int id, string message)
    {
        this.cache.AddOrUpdate(id, message, (i, s) => message);
    }

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

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

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

    public virtual void AddOrUpdate(int id, string message)
    {
        this.cache.AddOrUpdate(id, message, (i, s) => message);
    }

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

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

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

    public virtual void AddOrUpdate(int id, string message)
    {
        this.cache.AddOrUpdate(id, message, (i, s) => message);
    }

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

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

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

    public virtual void AddOrUpdate(int id, string message)
    {
        this.cache.AddOrUpdate(id, message, (i, s) => message);
    }

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

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

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

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

        this.cache.AddOrUpdate(id, message, (i, s) => message);
    }

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

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

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

    public virtual void AddOrUpdate(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);
    }
}
```

```
    this.Log.Saved(id);
}

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

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

protected virtual IStore Store
{
    this.Log.Saved(id);
}

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

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

protected virtual IStore Store
{
```

```
public class FileStore : IStore
{
    public FileStore(DirectoryInfo workingDirectory)

    public virtual void WriteAllText(int id, string message)

    public virtual Maybe<string> ReadAllText(int id)
    {
        var file = this.GetFileInfo(id);
        if (!file.Exists)
            return new Maybe<string>();
        var path = file.FullName;
        return new Maybe<string>(File.ReadAllText(path));
    }

    public virtual FileInfo GetFileInfo(int id)
    {
        return new FileInfo(
            Path.Combine(this.workingDirectory.FullName, id + ".txt"));
    }
}
```

```
public class FileStore : IStore
{
    public FileStore(DirectoryInfo workingDirectory)

    public virtual void WriteAllText(int id, string message)

    public virtual Maybe<string> ReadAllText(int id)
    {
        var file = this.GetFileInfo(id);
        if (!file.Exists)
            return new Maybe<string>();
        var path = file.FullName;
        return new Maybe<string>(File.ReadAllText(path));
    }

    public virtual FileInfo GetFileInfo(int id)
    {
        return new FileInfo(
            Path.Combine(this.workingDirectory.FullName, id + ".txt"));
    }
}
```

```csharp
public void Save(int id, string message)
{
    this.Log.Saving(id);
    this.Store.WriteAllText(id, message);
    this.Cache.AddOrUpdate(id, message);
    this.Log.Saved(id);
}

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.Store.GetFileInfo(id);
}
```

```csharp
public class SqlStore : IStore
{
    public void WriteAllText(int id, string message)
    {
        // Write to database here
    }

    public Maybe<string> ReadAllText(int id)
    {
        // Read and return from database here
    }

    public FileInfo GetFileInfo(int id)
    {
        throw new NotSupportedException();
    }
}
```

```
public void Save(int id, string message)
{
    this.Log.Saving(id);
    this.Store.WriteAllText(id, message);
    this.Cache.AddOrUpdate(id, message);
    this.Log.Saved(id);
}

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.Store.GetFileInfo(id);
}
```