SOLID Principles for C# Developers

SINGLE RESPONSIBILITY PRINCIPLE

Steve Smith
FORCE MULTIPLIER FOR DEV TEAMS
@ardalis | ardalis.com | weeklydevtips.com

SOLID

SRP - Single Responsibility Principle

OCP - Open Closed Principle

LSP - Liskov Substitution Principle

ISP - Interface Segregation Principle

DIP - Dependency Inversion Principle

SOLID Principles

SRP

Single Responsibility Principle

OCP

Open / Closed Principle

LSP

Liskov Substitution Principle

Interface Segregation Principle

ISP

Dependency Inversion Principle

DIP

Practice PDD

Pain Driven Development (PDD)

Avoid premature optimization

If current design is painful to work with, use principles to guide redesign

Single Responsibility Principle

Each software module should have one and only one reason to change.

The individual classes and methods in our applications define what the application does, and how it does it.

Multipurpose tools don't perform as well as dedicated tools

Dedicated tools are easier to use

A problem with one part of a multipurpose tool can impact all parts

What Is a Responsibility?

Persistence

Logging

Validation

Business Logic

Reasons to Change

Persistence

Logging

Validation

Business Logic

Responsibilities change
at different times
for different reasons.

Each one is an axis of change.

Axes of Change

Chief Information / Technology Officer

Chief Security Officer

Chief Operations Officer

Chief Marketing Officer

Tight Coupling

Binds two (or more) details together in a way that’s difficult to change.

Loose Coupling

Offers a modular way to choose which details are involved in a particular operation.

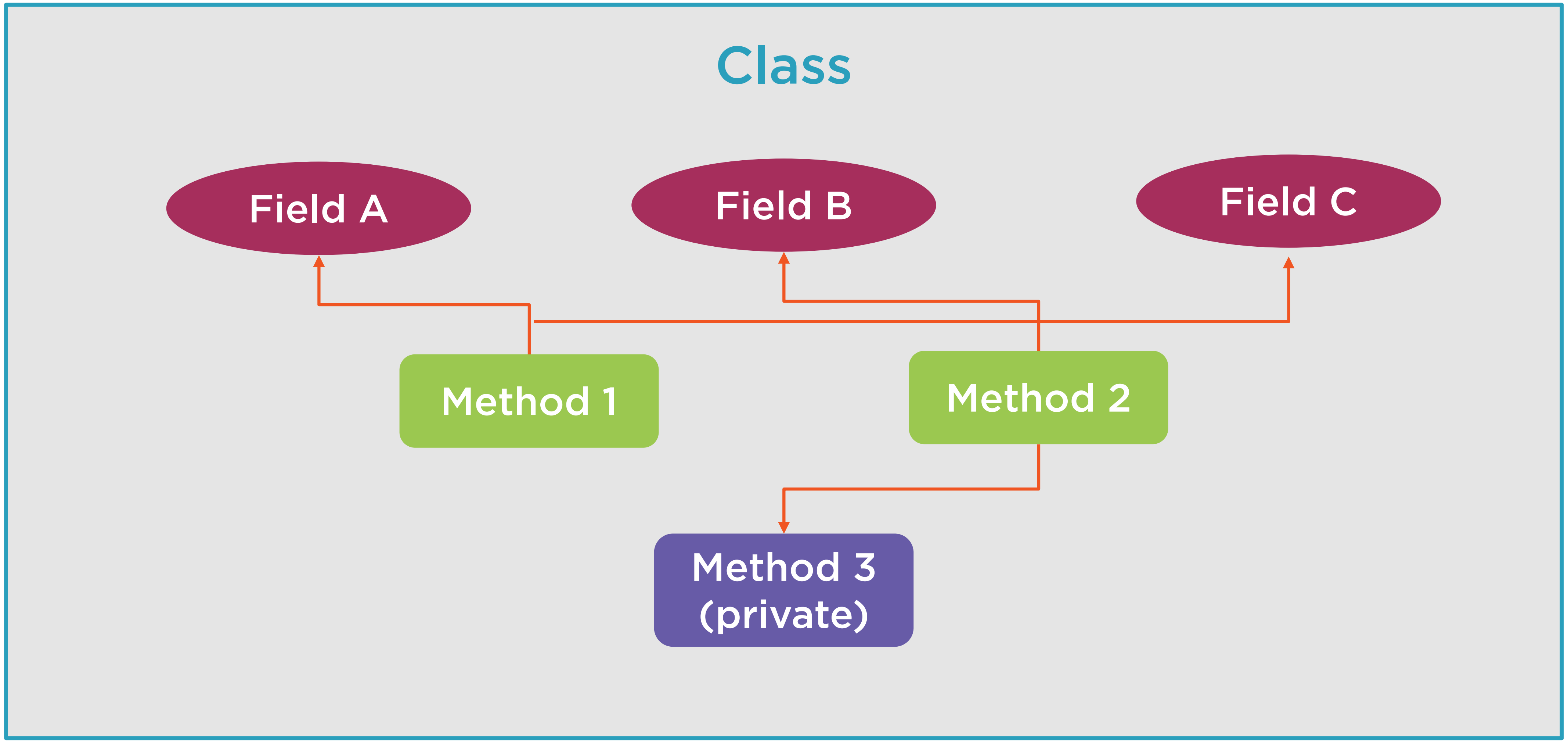
Separation of Concerns

Programs should be separated into distinct sections, each addressing a separate concern, or set of information that affects the program.

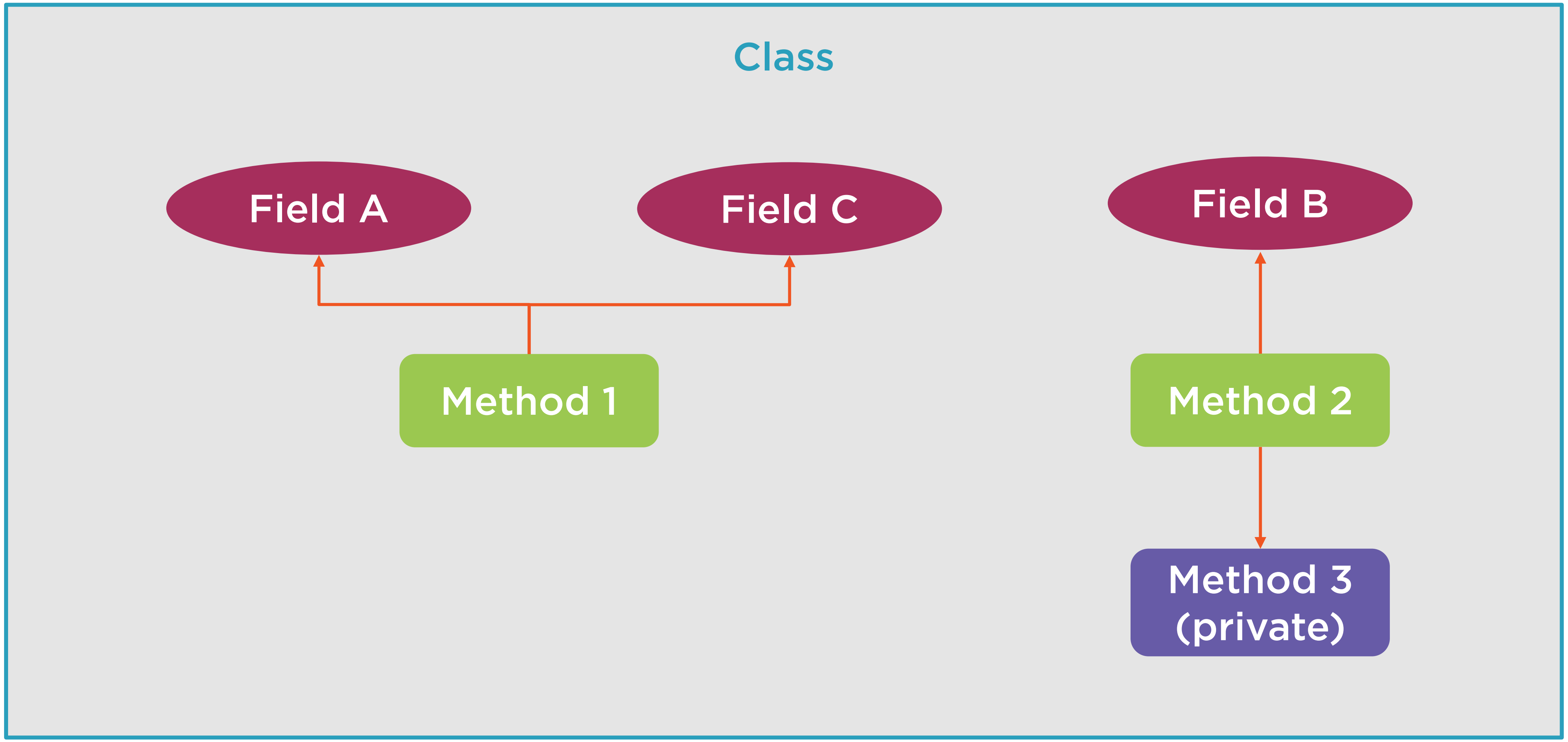
Keep plumbing code separate from high level business logic

Class elements that belong together are cohesive.

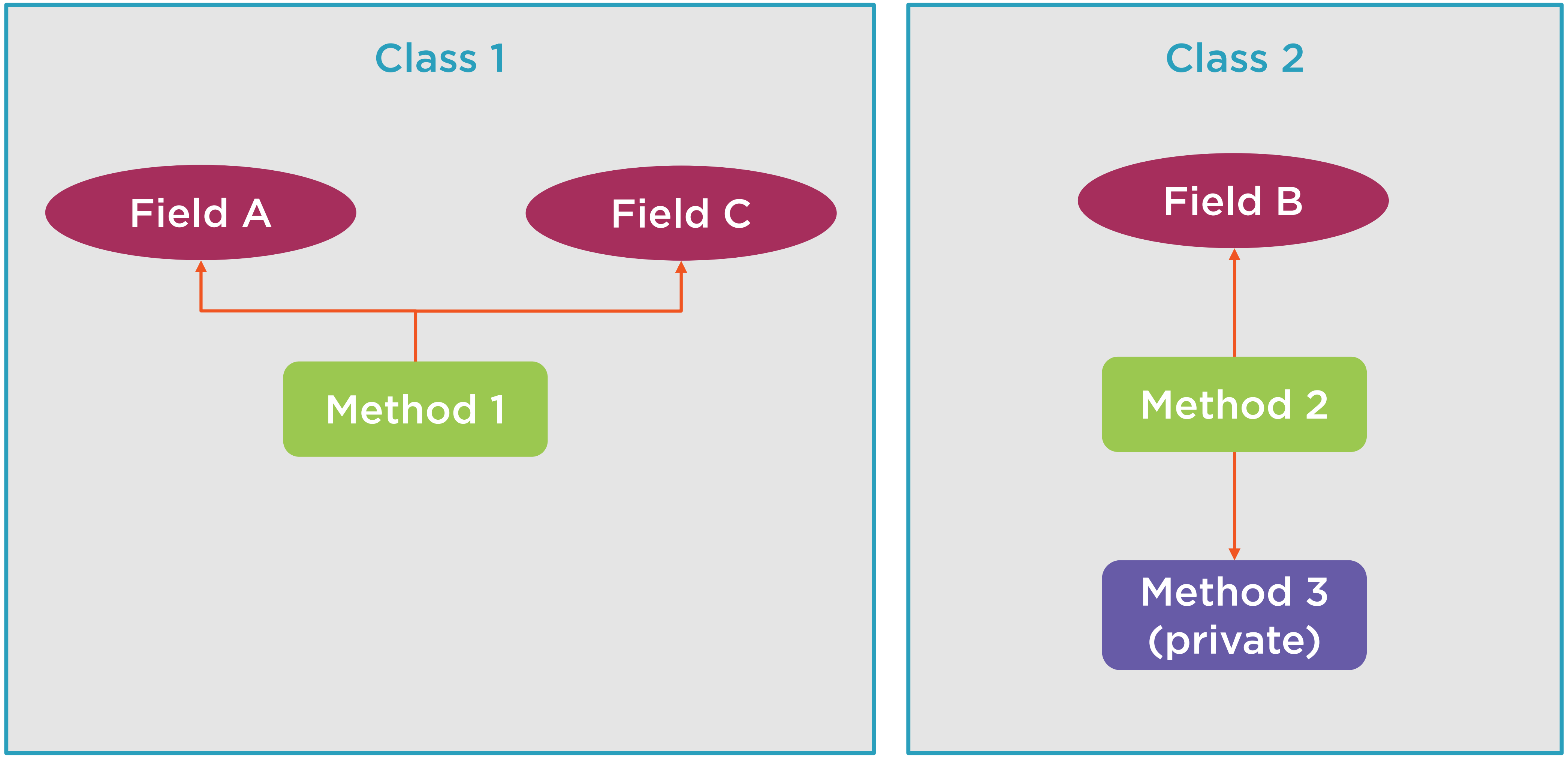
Class Cohesion



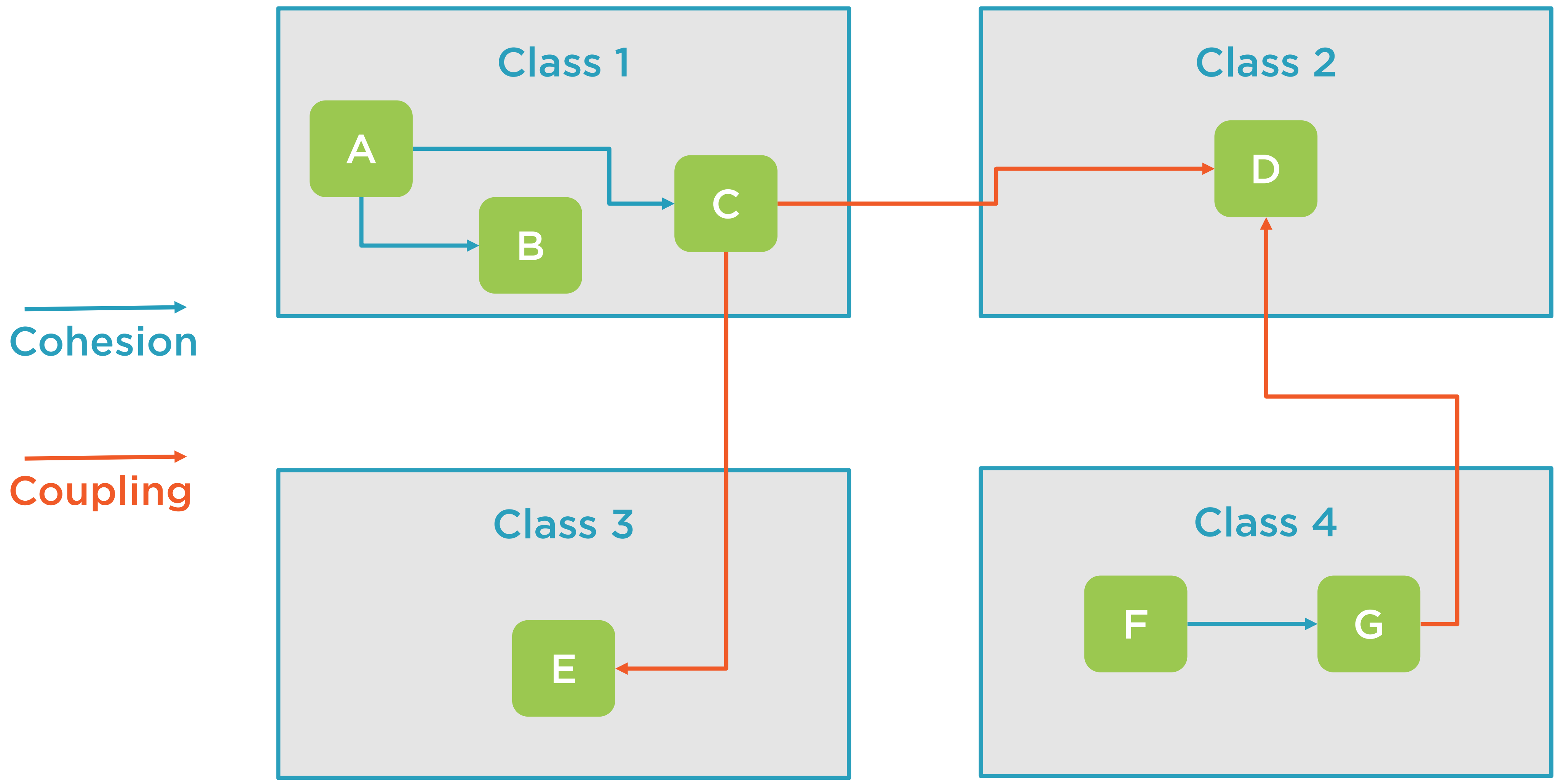
Class Cohesion (Low)



Class Cohesion (High)



Class Coupling and Cohesion



Demo

An Insurance Rating Service

Available at https://github.com/ardalis/solidsample

How many responsibilities did you find in `RatingEngine`?

```
Console.WriteLine("Starting
rate.");
```

◀ Logging

```
string policyJson =
File.ReadAllText("policy.json
");
```

◀ Persistence

```
var policy =
JsonConvert.DeserializeObject
<Policy>(policyJson,
new StringEnumConverter());
```

◀ Encoding Format

```
case PolicyType.Auto:
```

◂ Business Rule – Type of Policy (several of these)

```
if
(String.IsNullOrEmpty(policy.
Make))
```

◂ Validation (many examples)

```
int age = DateTime.Today.Year -
policy.DateOfBirth.Year;
if (policy.DateOfBirth.Month ==
DateTime.Today.Month &&
DateTime.Today.Day <
policy.DateOfBirth.Day ||
DateTime.Today.Month <
policy.DateOfBirth.Month)
{
    age--;
}
```

◂ Age Calculation

Responsibilities and Testability

Difficult to test one responsibility in isolation

Tests become

- Longer
- More complex
- Brittle
- Coupled to implementation

Testing

```
[Fact]
public void ReturnsRatingOf10000For200000LandPolicy()
{
    var policy = new Policy { Type = PolicyType.Land,
        BondAmount = 200000,Valuation = 200000 };
    string json = JsonConvert.SerializeObject(policy);
    File.WriteAllText("policy.json", json);

    var engine = new RatingEngine();
    engine.Rate();
    var result = engine.Rating;

    Assert.Equal(10000, result);
}
```

Applying SRP to `RatingEngine`

Logging

```
public class ConsoleLogger
{
    public void Log(string message)
    {
        Console.WriteLine(message);
    }
}
```

Logging

```
public class RatingEngine
{
    public ConsoleLogger Logger { get; set; } = new
ConsoleLogger();
    …
}
```

Logging

```
Console.WriteLine("Rating LAND policy...");
Console.WriteLine("Validating policy.");
if (policy.BondAmount == 0 || policy.Valuation == 0)
{
    Console.WriteLine("Land policy must specify Bond Amount
and Valuation.");
    return;
}
```

Logging

```
Logger.Log("Rating LAND policy...");
Logger.Log("Validating policy.");
if (policy.BondAmount == 0 || policy.Valuation == 0)
{
    Logger.Log("Land policy must specify Bond Amount and
Valuation.");
    return;
}
```

Persistence

```
public class FilePolicySource
{
    public string GetPolicyFromSource()
    {
        return File.ReadAllText("policy.json");
    }
}
```

Persistence

```
public class RatingEngine
{
    public FilePolicySource PolicySource { get; set; } =
new FilePolicySource();
    …
}
```

Persistence

```
string policyJson = PolicySource.GetPolicyFromSource();
```

```
public void Rate()

{

    Logger.Log("Starting rate.");
    Logger.Log("Loading policy.");


    string policyJson =
PolicySource.GetPolicyFromSource();


    var policy =
PolicySerializer.GetPolicyFromJsonS
tring(policyJson);


    …
}
```

- Logging (how is delegated)
- Persistence (how is delegated)
- Encoding Format (how is delegated)

Learning More

Pluralsight courses

- "Refactoring Fundamentals"
- "Microsoft Azure Developer: Refactoring Code"

Improved Testability

The 3 new classes are easily tested

The RatingEngine can now swap in test implementations of these 3 dependencies

Testing Serializer in Isolation

```
[Fact]
public void ReturnsDefaultPolicyFromEmptyJsonString()
{
    var inputJson = "{}";
    var serializer = new JsonPolicySerializer();

    var result = serializer.GetPolicyFromJsonString(inputJson);

    var policy = new Policy();
    AssertPoliciesEqual(result, policy);
}
```

SOLID Principles

Single Responsibility Principle ✓

Open / Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Key Takeaways

Practice Pain Driven Development!

Each class should have a single responsibility, or reason to change

Strive for high cohesion and loose coupling

Keep classes small, focused, and testable