# Understanding EF Encapsulation and the Great Repository Debates

**Julie Lerman**
MOST TRUSTED AUTHORITY ON ENTITY FRAMEWORK

@julielerman thedatafarm.com

# Module Overview

Benefits to Encapsulating EF Code

Reuse and Separation of Concerns

What About Repositories?

IQueryables or IEnumerables?

# Encapsulation

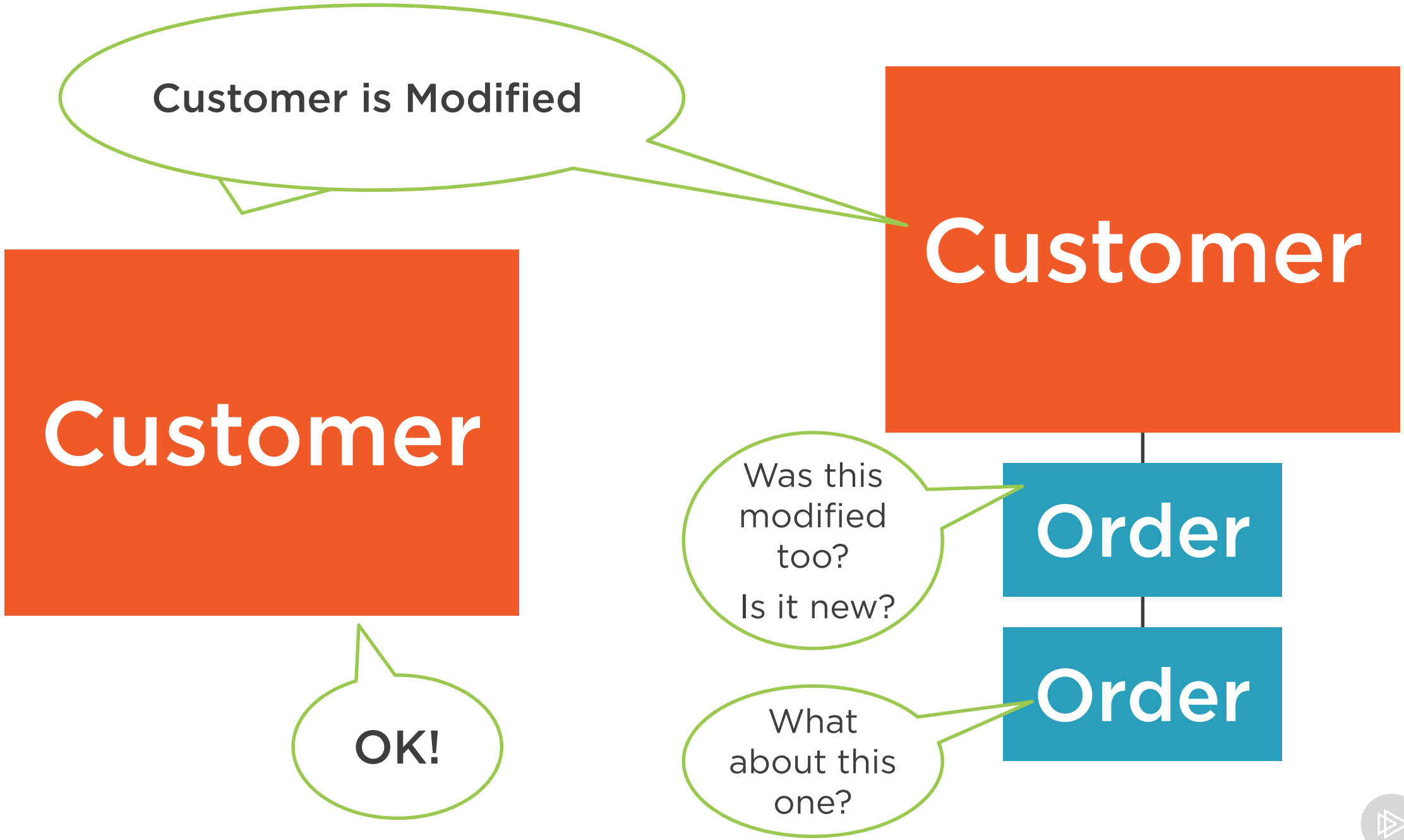
# Encapsulation in Object-Oriented Programming

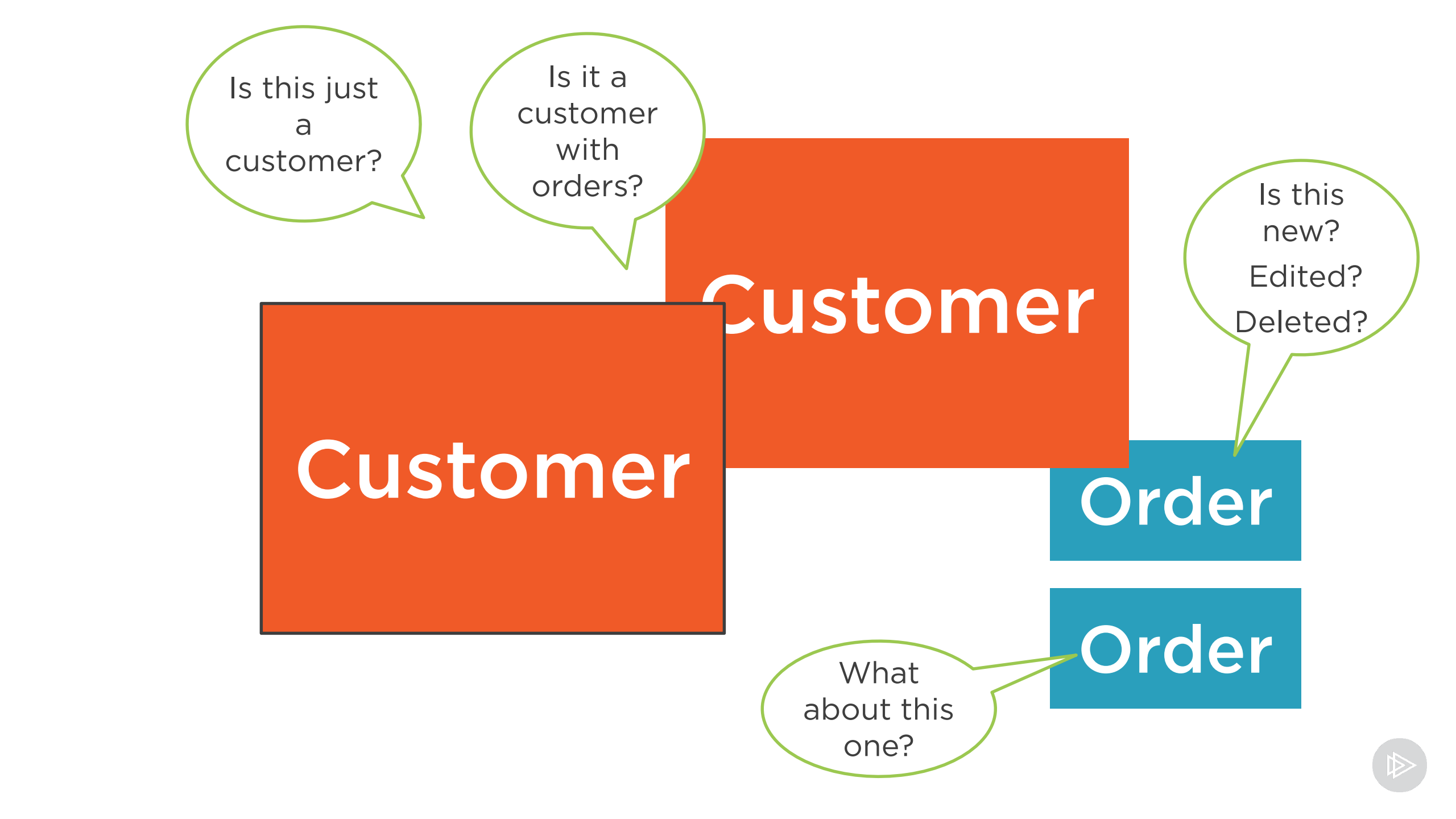
A technique for hiding data and implementation of a class providing accessors in the form of public methods and properties.

# Reusable Components


# But Don’t Overdo It!


# Separation of Concerns


Customer is Modified
Customer
Customer
OK!
Was this modified too?
Is it new?
Order
Order
What about this one?


Is this just a customer?
Is it a customer with orders?
Customer
Customer
Is this new?
Edited?
Deleted?
Order
Order
What about this one?

# Win

Building the Focused Class

# Win

Consuming the Focused Class

# Repository

# DbSet Is A Repository

## Entity Framework DbSet

***"represents the collection of all entities** [of a single type] **in the context, or that can be queried from the database"****

**DbSet.(LINQ query methods)**

**DbSet<Type>.Add/Attach**

**DbSet.Remove**

*msdn.microsoft.com/en-us/library/gg696460(v=vs.113).aspx

## Repository Pattern

***"in-memory domain object collection"****

**Execute Queries defined elsewhere**

**Add objects to repository**

**Remove objects from repository**

*martinfowler.com/eaaCatalog/repository.html

# Impact of DbSet Methods on Entities New to Change Tracker

| DbSet Method | Change Tracker State | SQL Command on SaveChanges |
|---|---|---|
| DbSet.Add | Added | INSERT |
| DbSet.Attach, *Set State=Modified* | Modified | UPDATE |
| DbSet.Attach | Unchanged | -- |
| DbSet.Remove | Deleted | DELETE |

# Repositories & EF: Pros, Cons & Observations

# Repositories Are Not Always the Answer

## Generic Repositories

**Simple Objects**
Graphs complicate things

**Straightforward CUD**
Create, Update, Delete

**Basic Queries**
GetAll (+Filter/Sort/Eager), Find By Id

**Predictable, Repeatable Patterns**
No special domain behavior required

## Explicit Persistence Classes

**Aggregates/Graphs**
Special handling of objects within graph

**Create, Update, Delete Not A Given**
Varying persistence rules/needs

**Explicit Querying**
Queries align with domain needs

**Behaviors Are Unique**
Needs Vary from Domain to Domain

# Baby-Step Repositories

```
public class CustomerRepository
{
  [_context setup code]

  public IQueryable<Customer> All {
    get { return _context.Customers; }
  }
  public Customer Find(int id) {
    return _context.Customers.Find(id);
  }
  public void Insert(Customer customer) {
    _context.Customers.Add(customer);
  }
  public void Update(Customer customer) {
    _context.Customers.Attach(customer);
    _context.Entry(customer).State = EntityState.Modified;
  }
  public void Delete(int id) {
    var customer = _context.Customers.Find(id);
    _context.Customers.Remove(customer);
  }
}
```

# Redundant Repositories

```
public class CustomerRepository
{
  {_context setup code}

  public IQueryable<Customer> All  {
    get { return _context.Customers; }
  }

  public Customer Find(int id) {
    return _context.Customers.Find(id);
  }

  public void Insert(Customer customer) {
    _context.Customers.Add(customer);
  }

  public void Update(Customer customer) {
    _context.Customers.Attach(customer);
    _context.Entry(customer).State =
                          EntityState.Modified;
  }

  public void Delete(int id) {
    var customer = _context.Customers.Find(id);
    _context.Customers.Remove(customer);
  }
}
```

```
public class OrderRepository
{
  {_context setup code}

  public IQueryable<Order> All  {
    get { return _context.Orders; }
  }

  public Order Find(int id) {
    return _context.Orders.Find(id);
  }

  public void Insert(Order order) {
    _context.Orders.Add(order);
  }

  public void Update(Order order) {
    _context.Orders.Attach(order);
    _context.Entry(order).State =
                          EntityState.Modified;
  }

  public void Delete(int id) {
    var  order  = Find(id);
    _context.Orders.Remove(order);
  }
}
```

```
public class ShipmentRepository
{
  {context setup code}

  public IQueryable<Shipment> All  {
    get { return _context.Shipments; }
  }

  public Shipment Find(int id) {
    return _context.Shipments.Find(id);
  }

  public void Insert(Shipment shipment) {
    _context.Shipments.Add(shipment);
  }

  public void Update(Shipment shipment) {
    _context.Shipments.Attach(shipment);
    _context.Entry(shipment).State =
                          EntityState.Modified;
  }

  public void Delete(int id) {
    var  order  = Find(id);
    _context.Shipments.Remove(shipment);
  }
}
```

# Repository Contract

```
public interface IEntityRepository<T> : IDisposable  {
  IQueryable<T> All { get; }
  T Find(int id);
  void Insert(T entity);
  void Update entity);
  void Delete(int id);
}
```

```
public class CustomerRepository : IEntityRepository
{
  {_context setup code}

  public IQueryable<Customer> All  {
    get { return _context.Customers; }
  }

  public Customer Find(int id) {
    return _context.Customers.Find(id);
  }

  public void Insert(Customer customer) {
    _context.Customers.Add(customer);
  }

  public void Update(Customer customer) {
    _context.Customers.Attach(customer);
    _context.Entry(customer).State =
                          EntityState.Modified;
  }

  public void Delete(int id) {
    var customer = _context.Customers.Find(id);
    _context.Customers.Remove(customer);
  }
}
```

```
public class OrderRepository : IEntityRepository
{
  {_context setup code}

  public IQueryable<Order> All  {
    get { return _context.Orders; }
  }

  public Order Find(int id) {
    return _context.Orders.Find(id);
  }

  public void Insert(Order order) {
    _context.Orders.Add(order);
  }

  public void Update(Order order) {
    _context.Orders.Attach(order);
    _context.Entry(order).State =
                          EntityState.Modified;
  }

  public void Delete(int id) {
    var  order  = Find(id);
    _context.Orders.Remove(order);
  }
}
```

```
public class ShipmentRepository : IEntityRepository
{
  {_context setup code}

  public IQueryable<Shipment> All  {
    get { return _context.Shipments; }
  }

  public Shipment Find(int id) {
    return _context.Shipments.Find(id);
  }

  public void Insert(Shipment shipment) {
    _context.Shipments.Add(shipment);
  }

  public void Update(Shipment shipment) {
    _context.Shipments.Attach(shipment);
    _context.Entry(shipment).State =
                          EntityState.Modified;
  }

  public void Delete(int id) {
    var  order  = Find(id);
    _context.Shipments.Remove(shipment);
  }
}
```

# Generic Repository (Simplified)

```
public class GenericRepository<TEntity> where TEntity : class {

    internal DbContext _context;
    internal DbSet<TEntity> _dbSet;

    public GenericRepository(DbContext context) {
      _context = context;
      _dbSet = context.Set<TEntity>();
    }

    public IQueryable<TEntity> All {
      IQueryable<TEntity> query = _dbSet;
      return = _dbSet;
    }

    public T Find(int id) {
      return _dbSet.Find(id);
    }

    public void Insert(TEntity entity) {
      _dbSet.Add(entity);
    }

    public void Update(TEntity  entity) {
      _dbSet.Attach(entity);
      _context.Entry(entity).State = EntityState.Modified;
    }

    public void Delete(int id) {
      var entity  = Find(id);
      _dbSet.Remove(entity);
    }
  }
```

# **Sage Advice:**
# Use a DbContext in Generic Repositories, Not An Abstraction

# Short-Lived DbContext Transactions

```
using (context=new SalesContext())
{
  return context.Orders.Where(o=>o.SubTotal>100);
}

using (context=new SalesContext())
{
   context.Orders.Add(someNewOrder);
   context.SaveChanges();
}
```

# Ongoing Transactions in Client-Side Apps

```
var context=new SalesContext())
var orders= context.Orders.Where(o=>o.Total>100).ToList();
orders[0].Total+=5;
orders[1].Total+=5;
orders.Add(new Order(…));
context.SaveChanges();
order2.ModifiedDate=DateTime.Today;
context.SaveChanges();
```

# Typical UOW with EF

```
public class UnitOfWork
{
  OrderSystemContext _context;
  public UOW() {
    _context = new OrderSystemContext();
  }
  public int Save() {
    return _context.SaveChanges();
  }
}
```

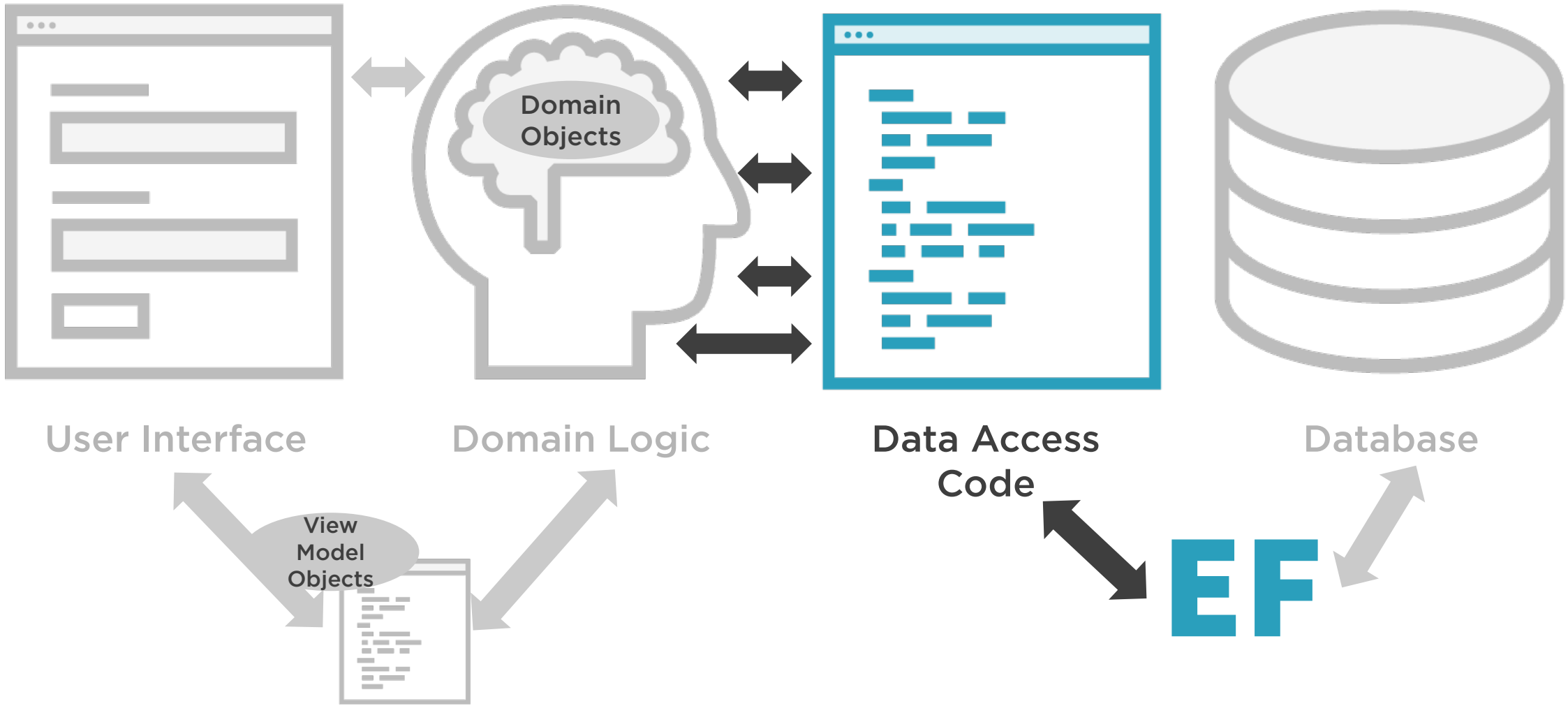
# UOW Encapsulating Generic EF

```
public class UnitOfWork {
   private GenericRepository<Customer> _custRepo;
   private GenericRepository<Order> _orderRepo;
   OrderSystemContext _context;

   public UnitOfWork () {
     _context = new OrderSystemContext();
   }

   public GenericRepository<Customer> CustomerRepository {
     get {
       if (_custRepo == null) {
         _custRepo = new GenericRepository<Customer>(_context);
      }
       return _custRepo;
     }
   }

   public GenericRepository<Order> OrderRepository { . . . }

   public GenericRepository<Detail> DetailRepository { . . . }

   public int Save() {
     return _context.SaveChanges();
   }
 }
```

IQueryable
or
IEnumerable?

**“It depends”**

# Encapsulated EF Code in Your Architecture



# Module Summary

Benefits to Encapsulating EF Code

Reuse and Separation of Concerns

What About Repositories?

IQueryables or IEnumerables?

# Resources

## Helpful Pluralsight Courses

SOLID Principles of Object Oriented Design bit.ly/1PhN6ny
Object-Oriented Programming Fundamentals in C# bit.ly/1RkzpYm
Encapsulation & Solid bit.ly/1TBYFcK

### For Repositories:

Implementing the Repository & Unit of Work Patterns in an ASP.NET MVC App bit.ly/1K8fCMo
Clean, Better, & Sexier Generic Repository Implementation for EF, Tugberk Ugurlu bit.ly/UD9sV6
EFReversePoco.com

### Against Repositories:

Repositories On Top UnitOfWork Are Not a Good Idea, Rob Conery: bit.ly/21cw0De
Favor query objects over repositories, Jimmy Bogard: bit.ly/1SQqhy5
Limiting your abstractions, Jimmy Bogard bit.ly/1KH8Ptm

Ayende.com/blog: Many good repository posts
martinfowler.com/eaaCatalog/unitOfWork.html

*Thanks to 100proofpress.com for permission to use my Queen of Hearts stamp in this video*

# Understanding EF Encapsulation and the Great Repository Debates

**Julie Lerman**
MOST TRUSTED AUTHORITY ON ENTITY FRAMEWORK

@julielerman thedatafarm.com