Implementing the Specification Pattern the Naive Way

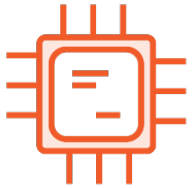
Vladimir Khorikov
PROGRAMMER

@vkhorikov www.enterprisecraftsmanship.com

How LINQ Works

```
MpaaRating <= MpaaRating.PG
```

```
if (MpaaRating <= MpaaRating.PG)
{
    /* ... */
}
```



Executed by processor

```
return session.Query<Movie>()
    .Where(x => x.MpaaRating <= MpaaRating.PG)
    .ToList();
```

Converted into SQL

How LINQ Works

IEnumerable

```
new [] { 1, 2 }.Where(x => x == 1)
```

```
public static IEnumerable<TSource> Where<TSource>(
    this IEnumerable<TSource> source,
    Func<TSource, bool> predicate);
```

```
Func<int, bool> func = x => x == 1;
```

IQueryable

```
dbContext.Movies
session.Query<Movie>()
    .Where(x =>
        x.MpaaRating <= MpaaRating.PG)
```

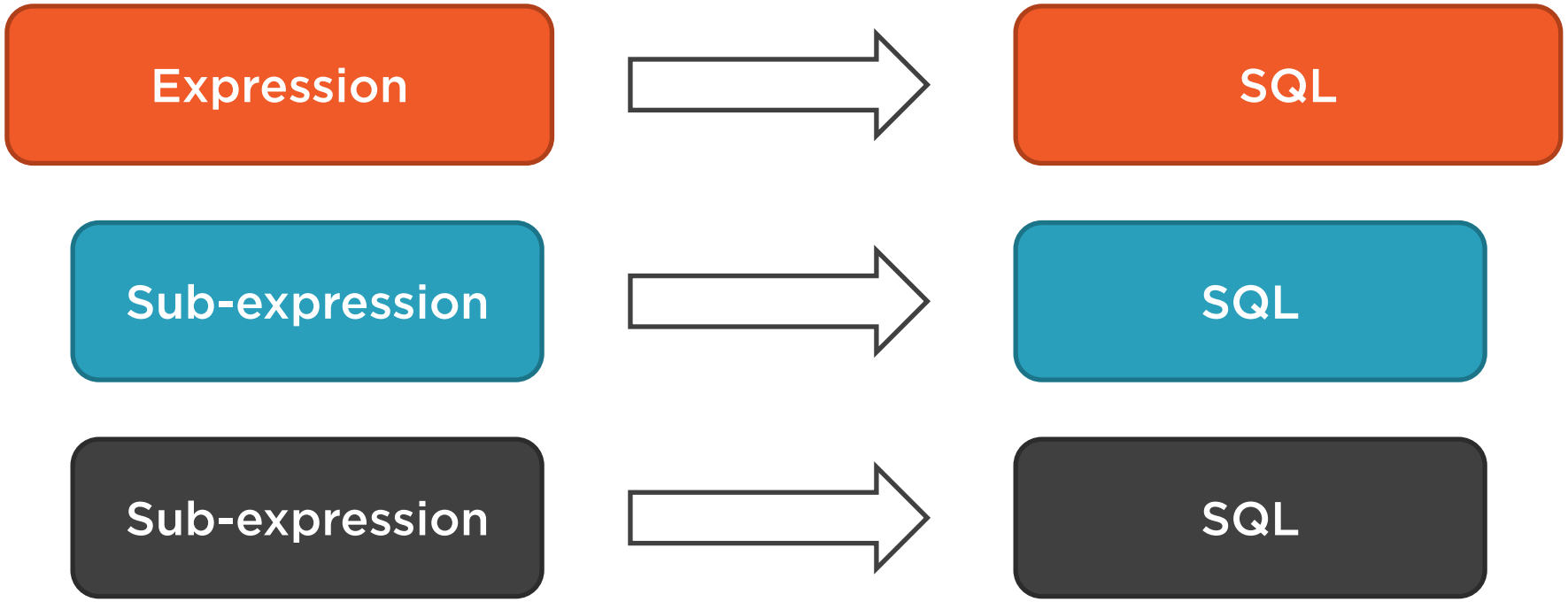
```
public static IQueryable<TSource> Where<TSource>(
    this IQueryable<TSource> source,
    Expression<Func<TSource, bool>> predicate);
```

```
Expression<Func<int, bool>> expression =
    x => x == 1;
```

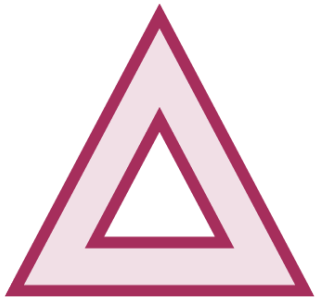
How LINQ Works

IQueryable : IEnumerable

How LINQ Works



How LINQ Works



Only a small set of operations can be translated into SQL

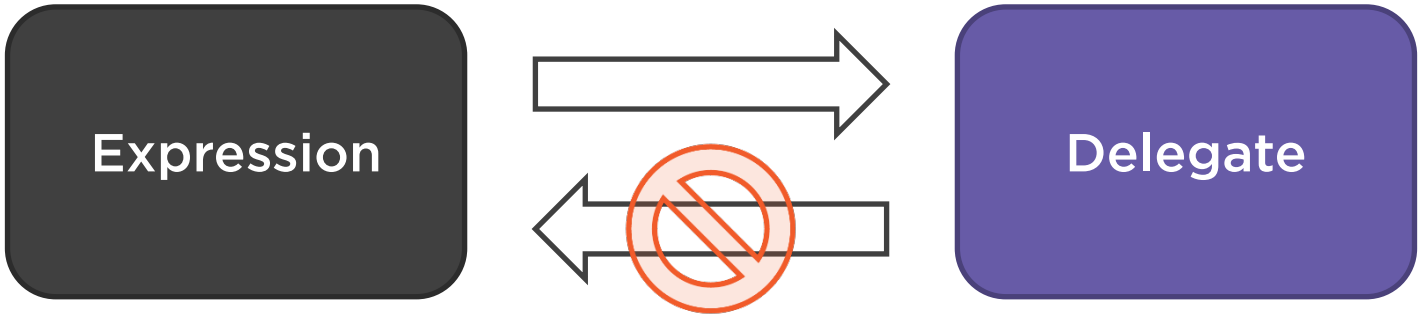
How LINQ Works

```
return session.Query<Movie>()
    .Where(x => x.IsSuitableForChildren())
    .ToList();
```

Exception

```
return session.Query<Movie>()
    .ToList()
    .Where(x => x.IsSuitableForChildren())
    .ToList();
```

How LINQ Works




```
Expression<Func<int, bool>> expression = x => x == 1;
Func<int, bool> func = expression.Compile();
```

Implementing the Specification Pattern the Naive Way

C# expressions

Recap: Using Plain C# Expressions

Removed duplication

Recap: Using Plain C# Expressions

```
public static readonly Expression<Func<Movie, bool>> IsSuitableForChildren =
    x => x.MpaaRating <= MpaaRating.PG;
public static readonly Expression<Func<Movie, bool>> HasCDVersion =
    x => x.ReleaseDate <= DateTime.Now.AddMonths(-6);
```

```
Expression<Func<Movie, bool>> exp1 = ForKidsOnly ? Movie.IsSuitableForChildren : x => true;
Expression<Func<Movie, bool>> exp2 = OnCD ? Movie.HasCDVersion : x => true;
Expression<Func<Movie, bool>> exp = exp1 && exp2; // Doesn't compile
```

```
Func<Movie, bool> isSuitableForChildren = Movie.IsSuitableForChildren.Compile();
if (!isSuitableForChildren(movie))
{
    /* ... */
}
```

✗ **Lack of encapsulation**

Implementing the Specification Pattern the Naive Way

~~C# expressions~~

Generic specifications

Recap: Using Generic Specifications

~~C# expressions~~

Generic specifications

Domain knowledge

Recap: Using Generic Specifications

~~C# expressions~~

~~Generic specifications~~

Domain knowledge

Recap: Using Generic Specifications

```
public class GenericSpecification<T>
{
    public Expression<Func<T, bool>> Expression { get; }

    public GenericSpecification(Expression<Func<T, bool>> expression)
    {
        Expression = expression;
    }

    public bool IsSatisfiedBy(T entity)
    {
        return Expression.Compile().Invoke(entity);
    }
}
```

Recap: Using Generic Specifications

Strongly typed specifications

KidsMovieSpecification

HasCDVersionSpecification

Returning IQueryable from a Repository

- Allows for arbitrary expressions, even unsupported ones
- Can cause runtime exceptions

- Could be fine for small projects
- Not suitable for medium and large projects

Returning IQueryable from a Repository

```
public IQueryable<Movie> Find()
```

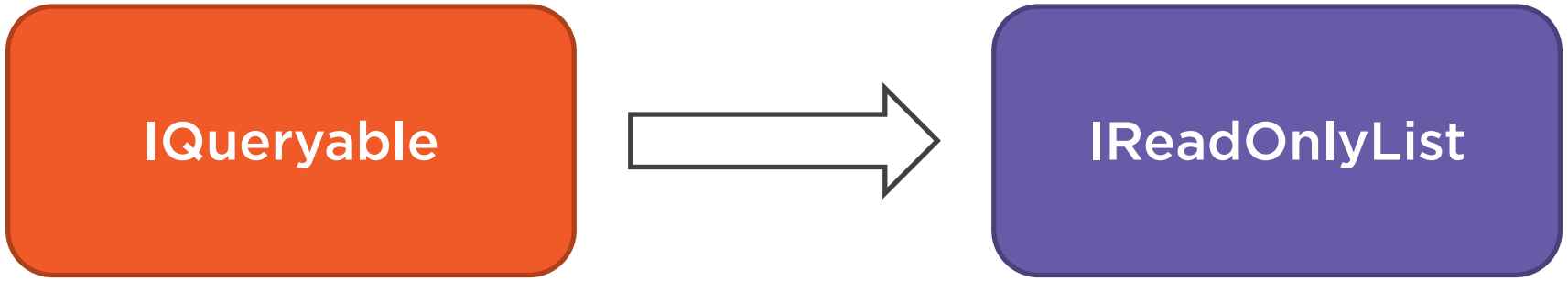
LSP violation

```
List<Movie> queryable = Find()
    .Where(x => x.CustomMethod1())
    .Where(x => x.CustomMethod2())
    .ToList();
```

Fails with an exception

Never return IQueryable
from your public methods

Returning IQueryable from a Repository



Returning IQueryable from a Repository

```
public IReadOnlyList<MovieDto> GetList(
    Specification<Movie> specification)
```

```
public IEnumerable<MovieDto> GetList(
    Specification<Movie> specification)
```

Returning IQueryable from a Repository

```
public IEnumerable<Movie> Find()
{
    using (ISession session = SessionFactory.OpenSession())
    {
        return session.Query<Movie>();
    }
}
```

✗ Exception

✗ LSP violation

Summary

How LINQ works

- IEnumerable and IQueryable
- C# lambda can compile to either a delegate or an expression
- C# expressions can compile to delegates

Using plain C# expressions

- Help get rid of duplication
- Don't provide proper encapsulation

Generic specifications

- A thin wrapper on top of expressions

Returning IQueryable from a repository

- Violates LSP

In the Next Module

Refactoring Towards Better Encapsulation