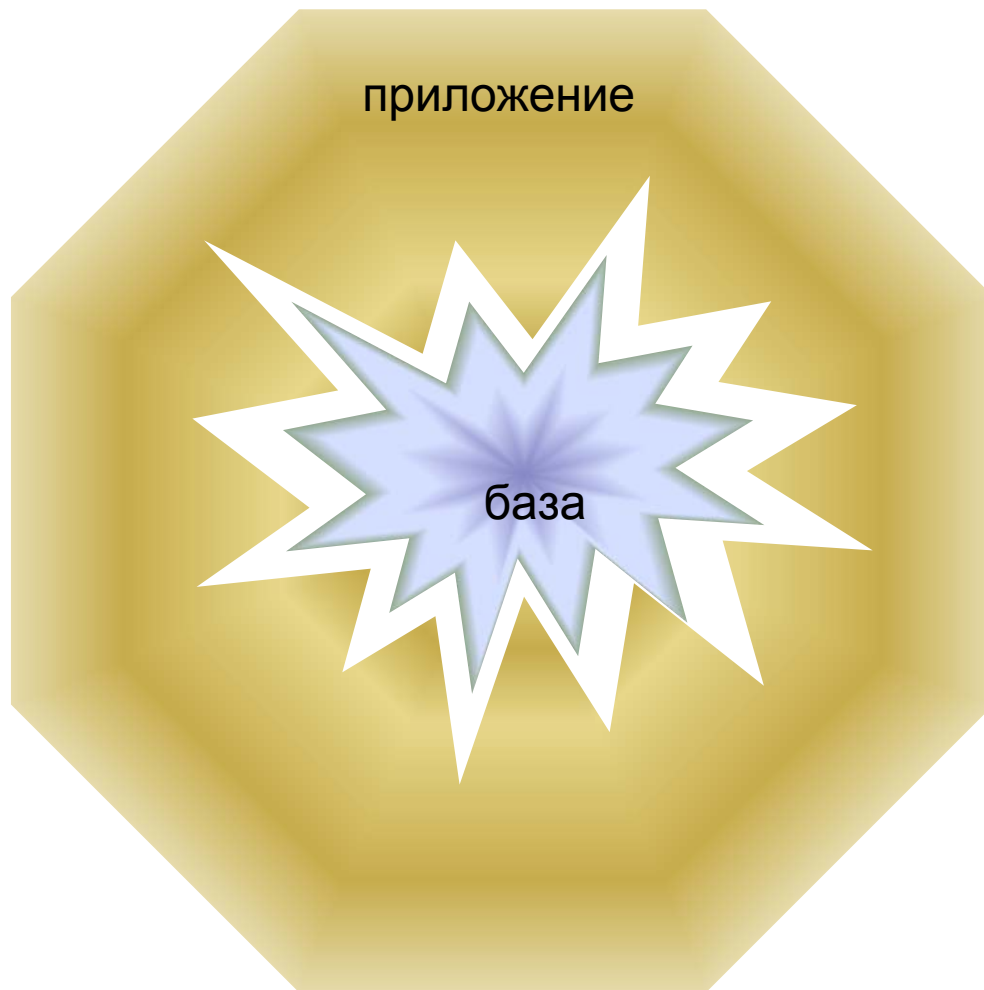


sm-art

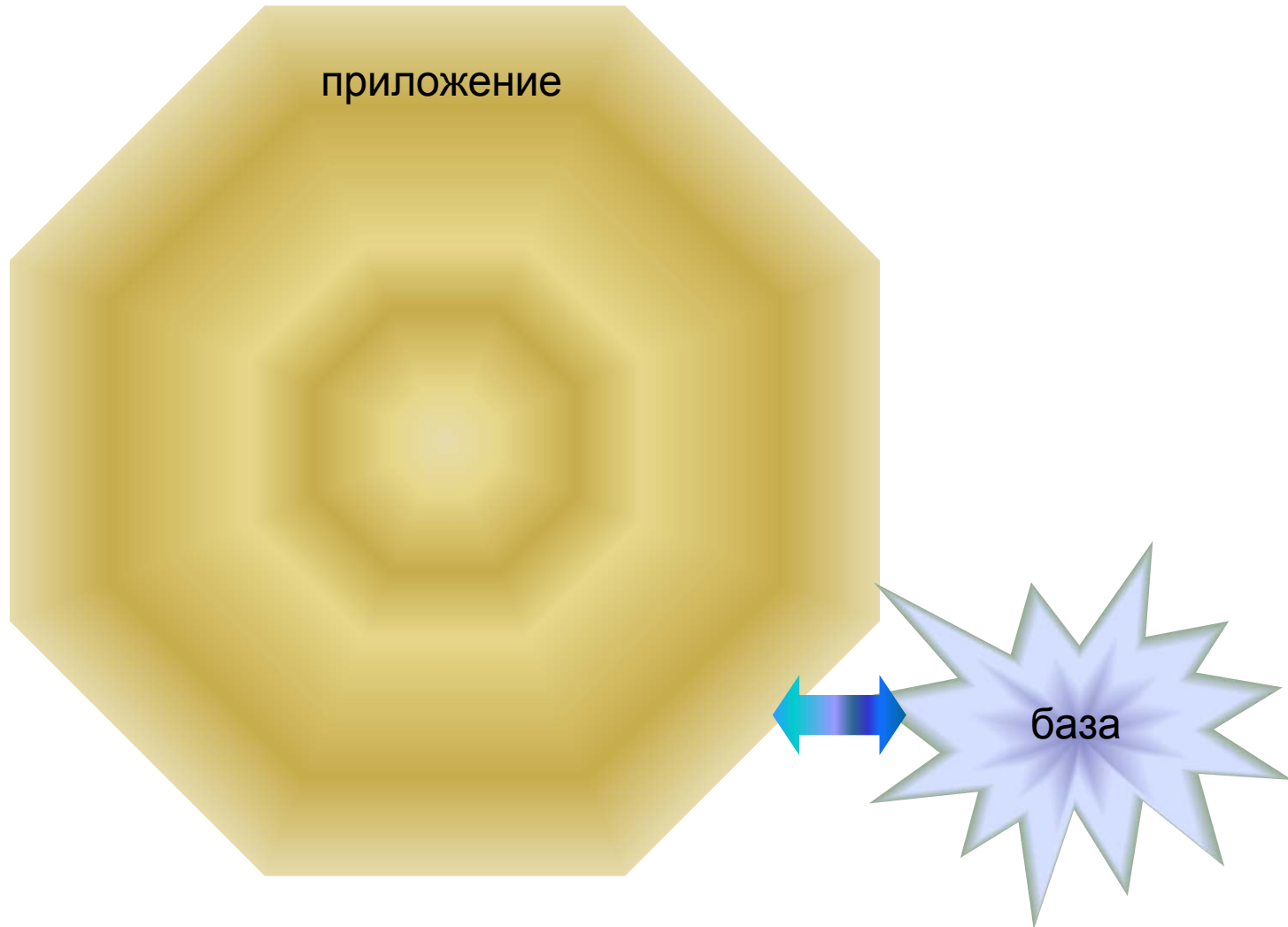
Паттерны доступа к данным

<http://sm-art.biz>
art@sm-art.biz

Database-centric



Persistence Ignorance



Active Record

```
public class Customer:
```

```
    ActiveRecordBase<Customer> {..}
```

```
Customer customer = new Customer() {..}
```

```
customer.Save();
```

Context-based

```
ISession session =  
    SessionFactory.CreateSession();  
session.Save(customer);
```

Полный PI???

- Виртуальные свойства
- Ленивая загрузка

Repository pattern

- DAL (не содержит бизнес-логики)
- Customer -> CustomerRepository
- Order, OrderItems -> OrderRepository

Методы

- Create, Update, Delete
- GetAllCustomers()
- GetCustomerById
- GetCustomersByNameAndAge и т.д.
- GetCustomersByTotalSales
- GetCustomersWithOutstandingDebtForLastQuarter
- GetCustomers(searchparams)

Generic Repository

```
public class RepositoryBase<TEntity> {  
    TEntity FetchById(object Id) {..};  
    IEnumerable<TEntity> FetchAll() {..};  
    IEnumerable<TEntity> Fetch(IQuery query){..};  
    void Save(TEntity entity) {..};  
    ...  
}
```

Проблемы

- Сортировка, постраничный вывод
- Повторяющийся код
- Бизнес-логика
- Трудности с тестированием
- Лишний слой абстракции

Specification Pattern

```
interface ISpecification<TEntity>{  
    bool IsSatisfiedBy(TEntity entity);  
}
```

КОМПОЗИЦИЯ СПЕЦИФИКАЦИЙ

```
public abstract class Specification : ISpecification
{
    public abstract bool Matches(T entity);

    public Specification And(ISpecification specification)
    {
        return new AndSpecification(this, specification);
    }

    public Specification Or(ISpecification specification)
    {
        return new OrSpecification(this, specification);
    }
}
```

Базовые спецификации

```
public class TotalOrdersSpecification : Specification
{
    private readonly decimal amount;
    internal TotalOrdersSpecification(decimal amount)
    {
        this.amount = amount;
    }
    public override bool Matches(Customer entity)
    {
        return (entity.TotalOrders > amount);
    }
}
```

Бизнес логика в спецификациях

```
var specification =  
    new TotalOrdersSpecification(42m).  
    And(new StatusSpecification(true));
```

Перегружаем операторы и получаем

```
var specification =  
    new TotalOrdersSpecification(42m) &  
    new StatusSpecification(true);
```

Проблемы

- Отбор производится простым перечислением
- Объекты должны быть вытащены из базы

Решение: необходим механизм трансляции спецификации в запрос (Query)

Query Object pattern

CustomersWithOutstandingDebtForLastQuarterQuery ->

-> SQL

-> IDbCommand

-> HQL

-> ICriteria

```
interface IQuery {  
    ICriteria GetQuery(ISession session);  
}
```

Использование

- Создаем в бизнес-слое
- Добавляем необходимые параметры из UI слоя (сортировку, разбивку по страницам..) и уровня приложения (eager loading)
- Исполняем

Repository + LINQ

```
interface IStorage <TEntity> {  
    IQueryable<TEntity> All();  
}  
  
class CustomerRepository {  
    private IStorage<Customer> _storage;  
    public IQueryable<Customer> GetCustomersByName (string  
        name) {  
        return _storage.All().Where(c => c.name == name);  
    }  
}
```

ИСТОЧНИК ДАННЫХ

```
public NHStorage<TEntity> : IStorage<TEntity> {  
    private ISession _session;  
    public IQueryable<TEntity> All() {  
        return _session.Linq<TEntity>();  
    }  
}
```

DlinqStorage

EFStorage

SubsonicStorage

В результате

- Абстрагировались от источника данных
- Легко заменить NHibernate на что-либо еще
- Для тестирования можно использовать in-memory storage
- В результате получаем IQueryable, с которым можно делать сортировку и т.д.

Query Object + LINQ

```
interface IQuery<TEntity> {  
    public Expression<Func<TEntity, bool>> AsExpression();  
}  
  
public class TopCustomersWithLowDiscountQuery {  
    public bool IncludePreferred { get; set; }  
    public decimal DiscountThreshold { get; set; }  
    public int SalesThreshold { get; set; }  
    public Expression<Func<Customer, bool>> AsExpression() {  
        return (c => c.Preferred == IncludePreferred &&  
            c.Discount < DiscountThreshold &&  
            c.AnnualSales > SalesThreshold);  
    }  
}
```

Как использовать

```
var query =  
    new TopCustomersWithLowDiscountQuery() {...};  
IQueryable<Customer> customers =  
    session.Linq<Customer>  
        .Where(query.AsExpression());
```

Можно упростить:

```
IQueryable<Customer> customers =  
    query.Fetch(session.Linq<Customer>);
```

Преимущества

- query не зависит от источника данных
- Единственная функция query – отбор
- Легко тестируется
- Возможна дальнейшая манипуляция данными до выполнения запроса к базе

Specification + LINQ

Было:

спес \Rightarrow System.Predicate<TEntity>

Стало:

спес \Rightarrow System.Linq.Expressions.Expression<System.Predicate<TEntity>>

- Содержит дерево выражений
- Может использоваться для фильтрации набора объектов
- Можно трансформировать в SQL и т.д.
- Годится для использования в Linq to ***

Заключение

- Repository не всегда удобно использовать
- Specification годится только для отбора готовых объектов
- Query Object может «собирать» из разных слоев приложения параметры выборки
- LINQ позволяет абстрагироваться от источника данных
- LINQ + Spec по возможности примерно равно LINQ + Query