VALUE OBJECTS, MICROTYPES UND DAS SPECIFICATION PATTERN

...DIE OFT UNTERSCHÄTZTEN BAUSTEINE VON DDD

Patrick Drechsler

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Wer praktiziert DDD*?

(*Domain Driven Design)

red·head·s
SOFTWARE SOLUTIONS SUPPORT

INHALT

- Value Objects
  - Was ist das?
  - Microtypes: Was ist das?
  - ORMs und Value Objects
  - Collections
- Specification Pattern: Business Regeln (auch zwischen Entitäten)

red·head·s
SOFTWARE SOLUTIONS SUPPORT

WAS IST EINE ENTITY?

- Id
- Lebenszyklus

WAS IST EIN VALUE OBJECT?

- Objekt ohne Id (immutable)
- hat attributbasierte Vergleichbarkeit
- ist oft "Cohesive" (verbindet z.B. Wert und Einheit)

red·head·s
SOFTWARE SOLUTIONS SUPPORT

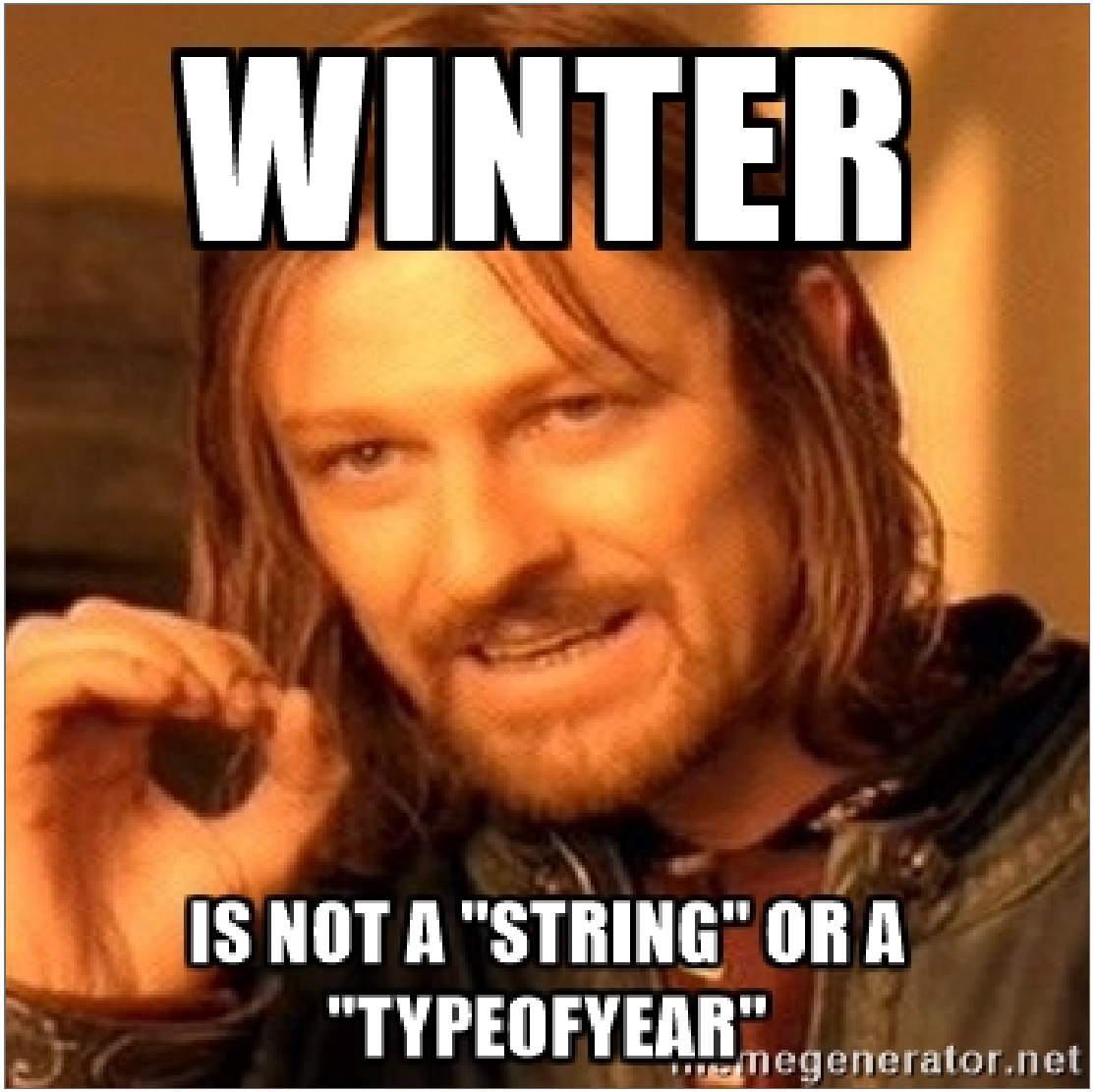
WIE ERKENNT MAN VALUE OBJECTS?

red·head·s
SOFTWARE SOLUTIONS SUPPORT

TYPISCHE KANDIDATEN

- String-Werte, die für die Domäne von Bedeutung sind
  - IP Adresse
  - IBAN
- Kombinationen wie
  - Betrag und Währung (42,13 EUR)
  - Wert und Einheit (23.4 kg)
- Adresse

red·head·s
SOFTWARE SOLUTIONS SUPPORT



red·head·s
SOFTWARE SOLUTIONS SUPPORT

...hat wahrscheinlich jeder schon mal gesehen:

```
public class Customer
{
    public int Id { get; set; } // evtl. in einer Entity-BaseClass
    //...
    public string EMailAddress { get; set; }
}
```

Problem:

- Datentyp 'string' passt nicht wirklich zu EMail Adresse.

red·head·s
SOFTWARE SOLUTIONS SUPPORT

```
public class Customer
{
    //...
    public EMailAddress EMailAddress { get; set; }
}
```

```
public class EMailAddress
{
    public EMailAddress(string value)
    {
        if (!IsValidEmailAddress(value))
            throw new MyInvalidEMailAddressException(value);

        Value = value;
    }

    public string Value { get; }

    private bool IsValidEmailAddress(string value)
    {
        try
        {
            new System.Net.Mail.MailAddress(value).Address = value;
        }
        catch { return false; }
    }
}
```

```
[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_
`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9]
(?:[a-z0-9-]*[a-z0-9])?
```

Value Object "zu Fuß"

```
public class EMailAddress
{
    public EMailAddress(string value)
    {
        if (!IsValidEmailAddress(value)) {
            /* throw */
        }

        Value = value;
    }

    public string Value { get; }

    private bool IsValidEmailAddress(string value) {
        // ...
    }
}
```

- Kein Default-Konstruktor
- Kein public setter fuer Value

red·head·s
SOFTWARE SOLUTIONS SUPPORT

```
public class Customer
{
    //...
    public EMailAddress EMailAddress { get; private set; }
}
```

- Die Klasse Customer kann nur gültige Email Adresse haben
- Das klärt nicht die Frage, ob eine Email für den Customer verpflichtend ist (dazu später mehr beim Specification Pattern)

red·head·s
SOFTWARE SOLUTIONS SUPPORT

```
// within some service class
public void DoSomething(EMailAddress mailOrig, EMailAddress mailNew)
{
    if (mailOrig.Equals(mailNew))
    {
        // do foo...
    }
    else
    {
        // do bar...
    }

    if (mailOrig == mailNew)
    {
        // do baz...
    }
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

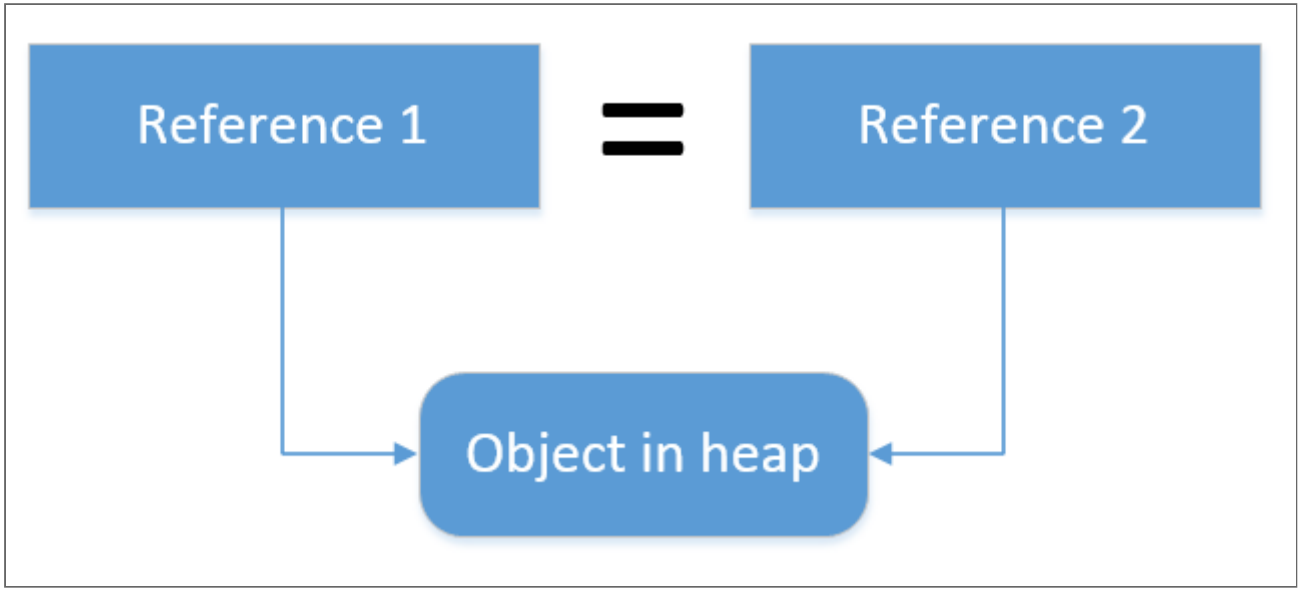
VERGLEICHBARKEIT IST ATTRIBUTBASIERT

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Exkurs Vergleichbarkeit

red·head·s
SOFTWARE SOLUTIONS SUPPORT

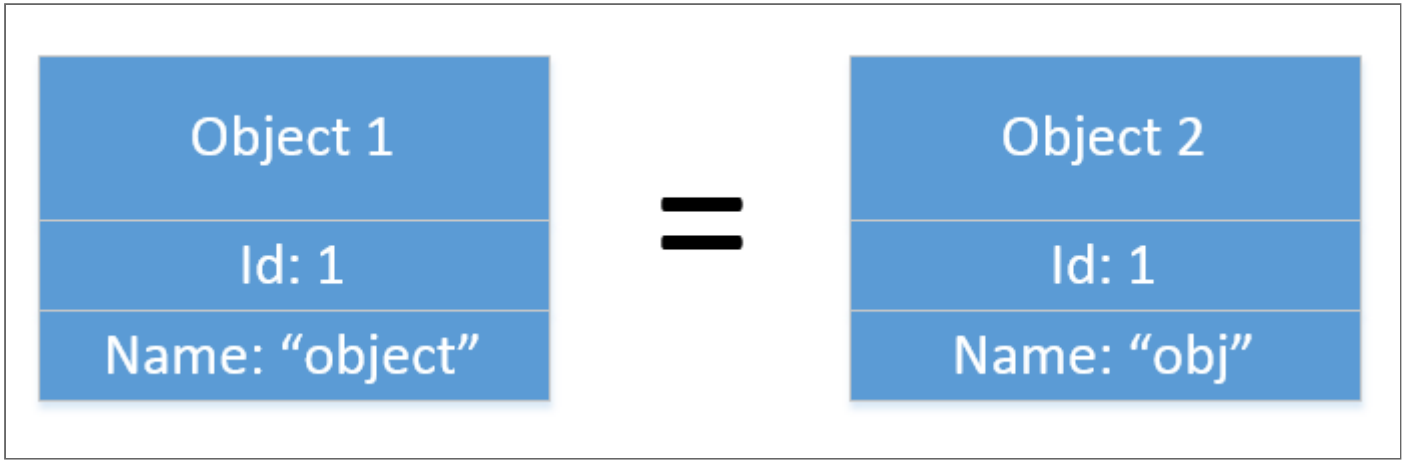
Equality by reference



http://enterprisecraftsmanship.com/2016/01/11/entity-vs-value-object-the-ultimate-list-of-differences/

red·head·s
SOFTWARE SOLUTIONS SUPPORT

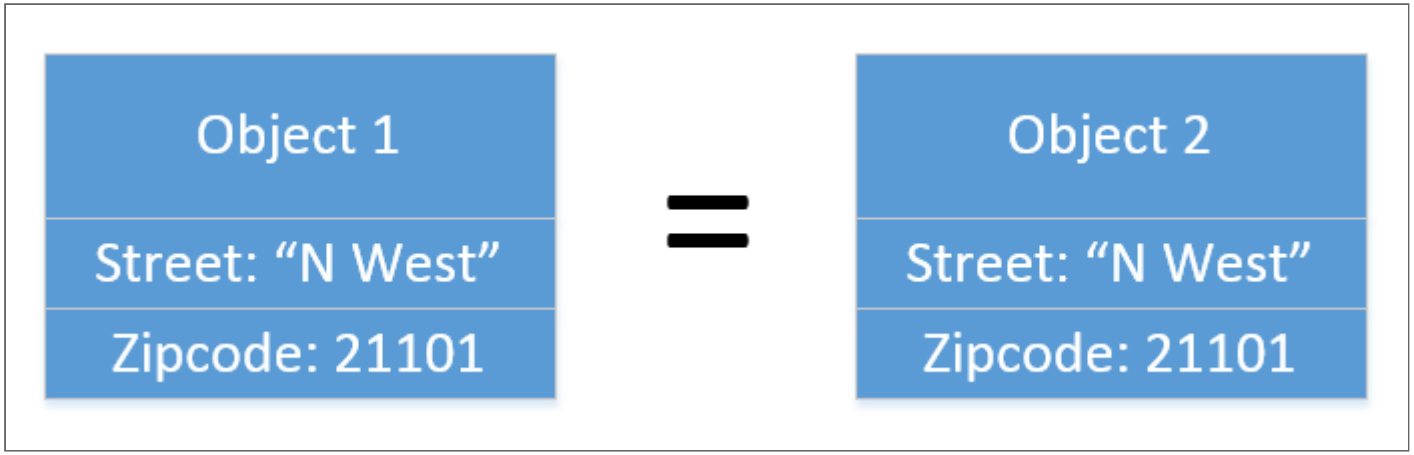
Equality by identifier



http://enterprisecraftsmanship.com/2016/01/11/entity-vs-value-object-the-ultimate-list-of-differences/

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Equality by structure



http://enterprisecraftsmanship.com/2016/01/11/entity-vs-value-object-the-ultimate-list-of-differences/

red·head·s
SOFTWARE SOLUTIONS SUPPORT

ERSTELLEN EINER VALUE OBJECT KLASSE (1/3)

```
public abstract class ValueObject<T> where T : ValueObject<T>
{
    protected abstract IEnumerable<object>
        GetAttributesToIncludeInEqualityCheck();
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

ERSTELLEN EINER VALUE OBJECT KLASSE (2/3)

```
public abstract class ValueObject<T> where T : ValueObject<T>
{
    protected abstract IEnumerable<object>
        GetAttributesToIncludeInEqualityCheck();

    public bool Equals(T other) {
        if (other == null) { return false }
        return GetAttributesToIncludeInEqualityCheck().
            .SequenceEqual(other.GetAttributesToIncludeInEqualityCheck());
    }

    public override int GetHashCode() {
        int hash = 17;
        foreach (var obj in this.GetAttributesToIncludeInEqualityCheck())
            hash = hash * 31 + (obj == null ? 0 : obj.GetHashCode());

        return hash;
    }
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

ERSTELLEN EINER VALUE OBJECT KLASSE (3/3)

```
public abstract class ValueObject<T> where T : ValueObject<T>
{
    protected abstract IEnumerable<object>
        GetAttributesToIncludeInEqualityCheck();

    public override bool Equals(object other) {
        return Equals(other as T);
    }

    public bool Equals(T other) {
        if (other == null) { return false }
        return GetAttributesToIncludeInEqualityCheck().
            .SequenceEqual(other.GetAttributesToIncludeInEqualityCheck());
    }

    public static bool operator ==(ValueObject<T> left, ValueObject<T> right) {
        return Equals(left, right);
    }

    public static bool operator !=(ValueObject<T> left, ValueObject<T> right) {
        return !(left == right);
    }

    public override int GetHashCode() {
        //...
    }
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

```
public class EMailAddress : ValueObject<EMAilAddress>
{
    public EMailAddress(string value)
    {
        if (!IsValidEmailAddress(value)) { /* throw */ }
        Value = value;
    }

    public string Value { get; }

    public override IEnumerable<object>
        GetAttributesToIncludeInEqualityCheck()
    {
        return new object[] { Value };
    }
    //...
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

VERGLEICHBARKEIT FÜR VALUE OBJECTS

Man überschreibt die `Equals` und `GetHashCode` Methoden, damit nur die Attribute (aka Properties) verglichen werden.

Bonus: Testen ist sehr einfach

red·head·s
SOFTWARE SOLUTIONS SUPPORT

FAZIT: VALUE OBJECTS

Immer dann, wenn

- der Basistyp (z.B. string) eigentlich ein Konzept der Domäne ist (z.B. IP Adresse, Zahl & Währung)
- die Property keinen Lebenszykus hat (z.B. Lieferadresse)

Value Objects sollten soviel Logik wie möglich beinhalten.

red·head·s
SOFTWARE SOLUTIONS SUPPORT

MICROTYPES

MICROTYPES

Mit Microtypes sind nicht etwa kleinere Einheiten von Value Objects gemeint (dachte ich ursprünglich).

MICROTYPES

- Ein Microtype ist eine **Erweiterung** eines bestehenden Value Objects.
- Aber nicht durch Ableitung, sondern durch Injection:

```
public class InternalEMailAddress
    : ValueObject<InternalEMailAddress> {

    public InternalEMailAddress(EMAilAddress value) {

        if (!IsInternalEMailAddress(value)) { /* throw */ }

        Value = value;
    }
}
```

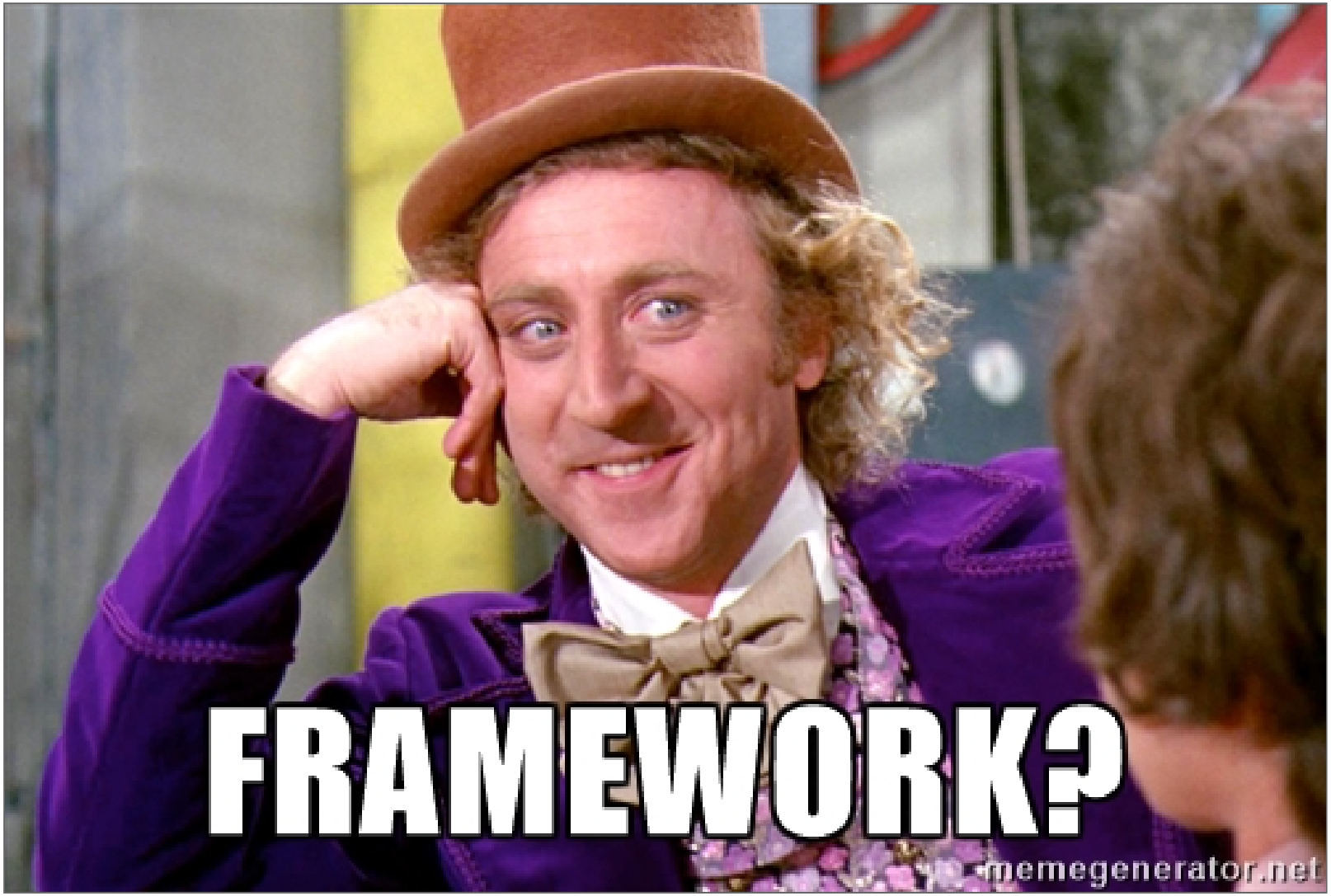
red·head·s
SOFTWARE SOLUTIONS SUPPORT

"Using microtypes is far from an industry standard. In fact, it's quite divisive. Some claim micro types are a precursor to clearer, more composable code, but for others, micro types are too many layers of annoying indirection. It's up to you to decide if you want to use the micro types pattern."

Scott Millet/Mick Tune in Patterns, Principles and Practices of Domain-Driven Design

red·head·s
SOFTWARE SOLUTIONS SUPPORT

FRAMEWORKS UND VALUE OBJECTS



red·head·s
SOFTWARE SOLUTIONS SUPPORT

ORM (Entity Framework, Hibernate, etc)...

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Muss der setter fuer das Attribut wirklich `public` sein?

Genuegt nicht

- `internal` oder
- `protected`?

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Oder noch besser:

Kann man die Klasse als Value Object beim ORM registrieren?

Ja

(NHibernate und Entity Framework)

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Bsp. Entity Framework

```
public class MyDbContext
{
    //...
    protected override void OnModelCreating(DbModelBuilder mb)
    {
        mb.ComplexType<EMailAddress>(); // <-- Value Object
    }

    public DbSet<Customer> Customers { get; set; } // <-- Entity
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Alternative: ORM nicht verwenden

(gerade im Umfeld von CQRS und Eventsourcing)

FAZIT: VALUE OBJECTS UND FRAMEWORKS

- Eigenschaften der Sprache nutzen (Sichtbarkeit von Settern)
- Dokumentation des Frameworks konsultieren
- Auf Framework verzichten

WIE SPEICHERT MAN LISTEN VON VALUE OBJECTS?

DDD: Ist das wirklich eine Collection?

Kann man die Liste von EMail-Adressen nicht abbilden als:

- HomeEMailAddress
- WorkEMailAddress
- OtherEMailAddress

Fazit: Die Frage, wie man das technisch löst, wird erstmal umgangen.

red·head·s
SOFTWARE SOLUTIONS SUPPORT

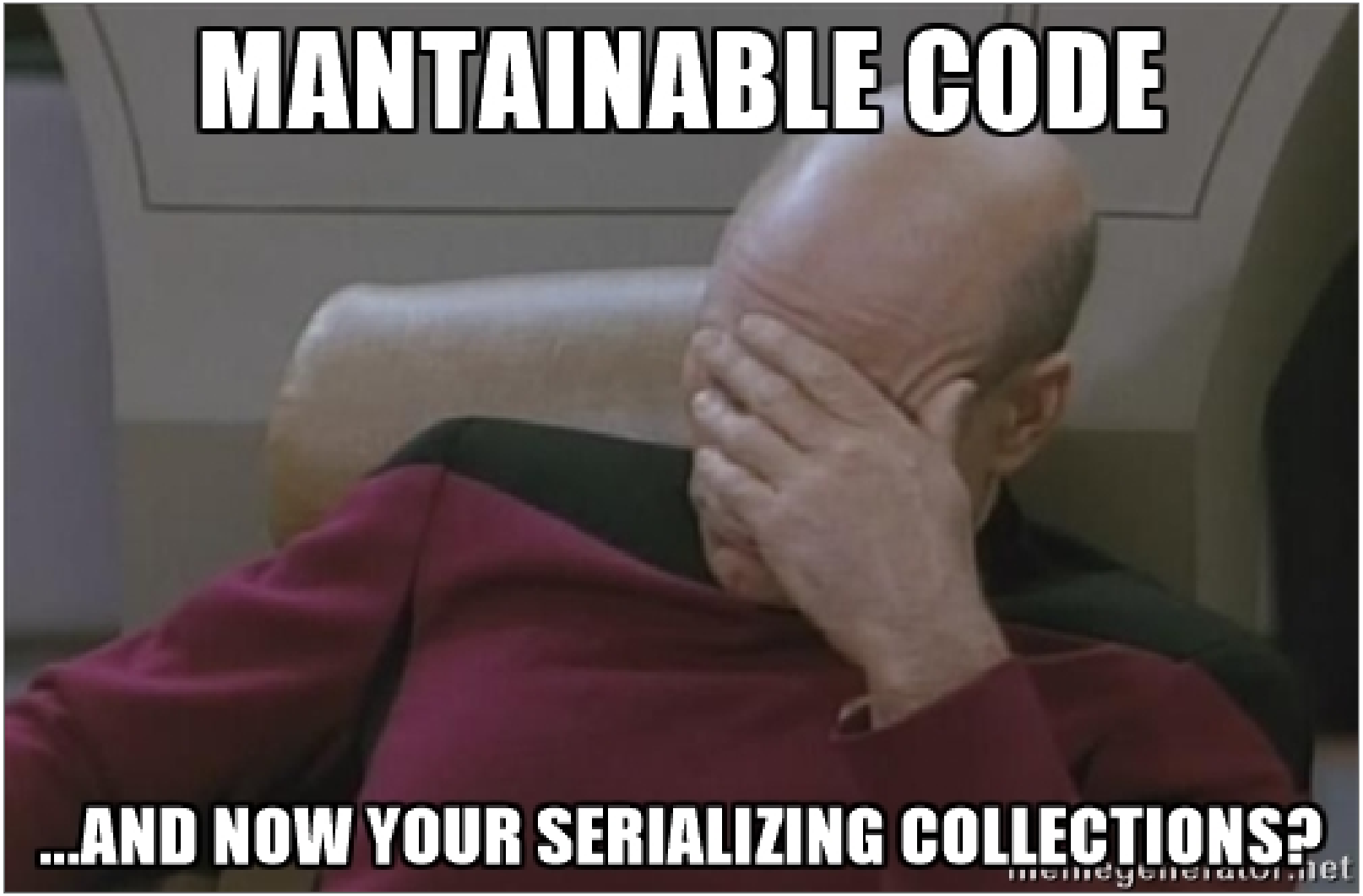
WIE SPEICHERT MAN LISTEN VON VALUE OBJECTS?

hier wird es interessant!

serialized string: JSON, XML, ...

(wenn man eine relationale DB verwendet)

red·head·s
SOFTWARE SOLUTIONS SUPPORT



red·head·s
SOFTWARE SOLUTIONS SUPPORT

DIE FRAGE SOLLTE SEIN:

- Wenn man eine **große** Collection von Value Objects hat:
  - Sind das dann wirklich noch Value Objects und nicht eher Entitäten?
  - Ist NoSQL besser geeignet (z.B. MongoDb)?
  - oder einfach nur Ids/URLs auf einen anderen Storage?
    - In-Memory (Redis)
    - Search Engine (Solar, Elastic)
    - andere Big Data Lösungen

red·head·s
SOFTWARE SOLUTIONS SUPPORT

FAZIT: VALUE OBJECTS UND COLLECTIONS

- Collections können serialisiert werden
- Große Collections:
  - in Entities umwandeln
  - auslagern

SPECIFICATION PATTERN

Business Regeln in eigene Objekte auslagern

red·head·s
SOFTWARE SOLUTIONS SUPPORT

"A specification pattern outlines a business rule ***that is combinable with other business rules****."*

https://en.wikipedia.org/wiki/Specification_pattern

red·head·s
SOFTWARE SOLUTIONS SUPPORT

"Specification pattern is a pattern that allows us to ***encapsulate some piece of domain knowledge into a single unit – specification – and reuse it in different parts of the code base.****"*

Vladimir Khorikov

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Also z.B. dem Customer Objekt

```
public class Customer
{
    //..
    public EMailAddress EMailAddress { get; set; }
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

- EMail ist ein Pflichtfeld
- Wie wärs mit einer **IsValid** Methode?

```
public class Customer
{
  //..
  public bool IsValid() {
      return EMAilAddress != null;
  }
}
```

Aber wenn wir diese Regel auch an anderer Stelle im Projekt brauchen?

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Extrahieren wir die IsValid Methode in eine eigene Klasse:

```
public class MandatoryStringSpecification : ISpecification<string>
{
    public bool IsSatisfiedBy(string s)
    {
        return !string.IsNullOrWhitespace(s);
    }
}

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

Bonus: Testbarkeit

red·head·s
SOFTWARE SOLUTIONS SUPPORT

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

public class MandatoryStringSpecification : ISpecification<string>
{
    public bool IsSatisfiedBy(string s)
    {
        return !string.IsNullOrWhitespace(s);
    }
}
```

```
public class Customer
{
    private MandatoryStringSpecification mandatoryString =
        new MandatoryStringSpecification();

    public bool IsValid()
    {
        if (!mandatoryString.IsSatisfiedBy(this.EMailAddress.Value))
        {
            throw new MyMandatoryFieldMissingException(nameof(this.EMailAddress));
        }
    }
}
```

Nett, aber das geht noch besser: Regeln können auch miteinander kombiniert werden.

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Beispiel von M. Fowler & E. Evans (**Wikipedia**):

- Wenn
  - Rechnung überfällig UND
  - Mahnung verschickt UND
  - Noch nicht bei Inkasso
- Dann
  - Beauftrage Inkasso mit Rechnung

red·head·s
SOFTWARE SOLUTIONS SUPPORT

```
// service class
var overDue = new OverDueSpecification();
var noticeSent = new NoticeSentSpecification();
var inCollection = new InCollectionSpecification();

// example of specification pattern logic chaining
var sendToCollection = overDue.And(noticeSent).And(inCollection.Not());

var invoiceCollection = anotherService.GetInvoices();

foreach (var currentInvoice in invoiceCollection) {
    if (sendToCollection.IsSatisfiedBy(currentInvoice))  {
        currentInvoice.SendToCollection();
    }
}
```

- Rechnung ("Invoice")
- Rechnung ist überfällig ("OverDue")
- Mahnung wurde verschickt ("NoticeSent")
- Noch nicht bei Inkasso ("InCollection.Not()")

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Wie ermöglicht man die Kombinierbarkeit der Specifications?

```
public interface ISpecification<T>
{
    bool IsSatisfiedBy(T entity);
    ISpecification<T> And(ISpecification<T> other);
    ISpecification<T> Not();
    //ISpecification<T> AndNot(ISpecification<T> other);
    //ISpecification<T> Or(ISpecification<T> other);
    //ISpecification<T> OrNot(ISpecification<T> other);
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

COMPOSITESPECIFICATION

```
public abstract class CompositeSpecification<T> : ISpecification<T>
{
    public abstract bool IsSatisfiedBy(T entity);

    public ISpecification<T> And(ISpecification<T> other)
    {
        return new AndSpecification<T>(this, other);
    }

    public ISpecification<T> Not()
    {
        return new NotSpecification<T>(this);
    }
    //...
}
```

- Rückgabewert ist immer vom Typ ISpecification
- Fluent API

red·head·s
SOFTWARE SOLUTIONS SUPPORT

ANDSPECIFICATION

```
public class AndSpecification<T> : CompositeSpecification<T>
{
    private readonly ISpecification<T> left;
    private readonly ISpecification<T> right;

    public AndSpecification(ISpecification<T> left, ISpecification<T> right)
    {
        this.left = left;
        this.right = right;
    }

    public override bool IsSatisfiedBy(T candidate)
    {
        return left.IsSatisfiedBy(candidate) && right.IsSatisfiedBy(candidate);
    }
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

NOTSPECIFICATION

```
public class NotSpecification<T> : CompositeSpecification<T>
{
    private readonly ISpecification<T> other;

    public NotSpecification(ISpecification<T> other)
    {
        this.other = other;
    }

    public override bool IsSatisfiedBy(T candidate)
    {
        return !other.IsSatisfiedBy(candidate);
    }
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Alternative Lösung ohne Specification Pattern:

```
//.. in the service class
var invoiceCollection = anotherService.GetInvoices();
foreach (Invoice currentInvoice in invoiceCollection)
{
    currentInvoice.SendToCollectionIfNecessary();
}

public class Invoice
{
    public bool ShouldSendToCollection
    {
        get
        {
            return this.OverDue && this.NoticeSent && this.InCollection == false;
        }
    }

    public void SendToCollectionIfNecessary()
    {
        if (!ShouldSendToCollection) return;
        this.SendToCollection();
    }
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Was ist mit Regeln, die **objektübergreifend** sind?

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Z.B.: Eine EMail darf nur einem Kunden zugewiesen sein

Anders ausgedrückt: Das Anlegen eines neuen Kunden soll unterbunden werden, wenn die EMail schon einem anderen Kunden zugewiesen ist.

Spätestens jetzt genügt das einzelne Kundenobjekt nicht mehr.

Wir müssen eine externe Quelle anzapfen.

Unsere Business Regel kann nicht mehr im Objekt selbst verankert sein.

Lsg: Unsere Service-Schicht verwendet eine neue UniqueEMailSpecification

red·head·s
SOFTWARE SOLUTIONS SUPPORT

```
public class Service
{
    private UniqueEMailSpecification _uniqueEmail; // more specifications go here

    public Service(IRepo repo) {
        _uniqueEmail = new UniqueEMailSpecification(repo);
    }

    public void CreateNewCustomer(Customer customer) {
        if (_uniqueEmail.IsSatisfiedBy(customer.EMail)) {
            // save new customer to repo
        } else { /* handle error case (ie throw) */ }
    }
}

public class UniqueEMailSpecification : CompositeSpecification<T>
{
    private IRepository _repo;

    public class UniqueEMailSpecification(IRepository repo) {
        _repo = repo;
    }

    public override bool IsSatisfiedBy(string mail) {
        return !repo.Customers.Contains(x => x.EMail.Equals(mail));
    }
}
```

red·head·s
SOFTWARE SOLUTIONS SUPPORT

FAZIT: SPECIFICATION PATTERN

V.a. dann nützlich, wenn die Regeln

- an mehreren Stellen verwendet werden und
- kombinierbar sein sollen

red·head·s
SOFTWARE SOLUTIONS SUPPORT

TAKE HOME MESSAGE

- **Value Objects**: für Objekte, die keinen eigenen Lebenszykus benötigen
- **Microtypes**: für granulare Value Objects
- **Specification Pattern**: für wiederverwendbare und kombinierbare Regeln

Alles auch ohne DDD einsetzbar

red·head·s
SOFTWARE SOLUTIONS SUPPORT

Beste Übersichtsseite: **https://github.com/heynickc/awesome-ddd**

- Bücher
  - E. Evans, Domain-Driven Design (the blue book)
  - V. Vernon, Implementing Domain-Driven Design (the red book)
  - S. Millet & N. Tune, Patterns, Principles and Practices of Domain-Driven Design
  - V. Vernon, Domain-Driven Design Distilled
- Blogs
  - **http://enterprisecraftsmanship.com/**
- Online trainings
  - **https://www.pluralsight.com/courses/domain-driven-design-in-practice**

red·head·s
SOFTWARE SOLUTIONS SUPPORT

DANKE

FRAGEN?