Power of Value –
Power Use of Value Objects in Domain Driven Design

QCon London 2009

Dan Bergh Johnsson
Partner and Spokesperson
Omegapoint AB, Sweden

omega point.

stolen

SmallTalk

Value Objects

omega point.

```
public class CustForm extends ActionForm {
    String phone;

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
}


public class AddCustAction extends Action {
    CustomerService custserv = null;


    @Override
    public ActionForward execute(
      ActionMapping actionMapping,
      ActionForm actionForm,
      HttpServletRequest httpServletRequest,
      HttpServletResponse httpServletResponse) {


        CustForm form = (CustForm) actionForm;
         try {
             String phone = form.getPhone();
             custserv.addCust(0, phone, "foo");
             return actionMapping.
                        findForward("success");
         } catch (ValidationException e) {
             return actionMapping.
                        findForward("invaliddata");
         }
    }
}
```

```
public interface CustomerService {
    void addCust(int i, String phone, String s)
            throws ValidationException;
}



public class CustomerServiceImpl
        implements CustomerService {

    public void addCust(int i,
                        String phone,
                        String s)
            throws ValidationException {

        PreparedStatement dbstmt = "INSERT …";

        if (!justnumbers(phone))
            throw new ValidationException();
        try {
            dbstmt.setInt(1, i);
            dbstmt.setString(3, phone);
            dbstmt.setString(4, s);
            dbstmt.executeUpdate();

        } catch (SQLException e) {
            throw new RuntimeException();
        }
    }
    static boolean justnumbers(String s) {
      return s.matches("[0-9]*");
    }
}
```

omega point.

```
void onlineTransaction(StoreId store, BigDecimal amount) {
       Currency storeCurrency = storeService.getCurrency(store);
       if (storeCurrency.equals(cardcurrency)) {
           debt = debt.add(amount);
       } else if (cardcurrency.equals(ExchangeService.REF_CURR) &&
                (!storeCurrency.equals(ExchangeService.REF_CURR))){
           QuoteDTO storequote =
        exchange.findCurrentRate(storeCurrency);
       debt = debt.add(amount.multiply(storequote.rate))
        .add(ExchangeService.FEE);
       } else if (!cardcurrency.equals(ExchangeService.REF_CURR) &&
                (storeCurrency.equals(ExchangeService.REF_CURR))){
           QuoteDTO cardquote = exchange.findCurrentRate(cardcurrency);
           debt = debt.add(amount.divide(cardquote.rate))
        .add(ExchangeService.FEE);
       } else {
           QuoteDTO cardquote = exchange.findCurrentRate(cardcurrency);
           QuoteDTO storequote =exchange.findCurrentRate(storeCurrency);
           debt = debt.add(amount.divide(cardquote.rate)
            .multiply(storequote.rate))
                   .add(ExchangeService.FEE.multiply(BigDecimal.valueOf(2)));
       }
   }
}
```

Overall Presentation Goal

Show how some power use of Value Objects can radically change design and code,

hopefully to the better

omega point.

Analysis / Conclusions

- Computation complexity moved to value objects
- Compound value objects can swallow lots of computational complexity
- Entities relieved of complexity
- Improved extensibility, esp testability and concurrency issues

omega point.

Warming Up

Simple Value Objects

DDD-style

omega point.

```
public class CustForm extends ActionForm {
    String phone;
    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
}


public class AddCustAction extends Action {
    CustomerService custserv = null;

    @Override
    public ActionForward execute(
      ActionMapping actionMapping,
      ActionForm actionForm,
      HttpServletRequest httpServletRequest,
      HttpServletResponse httpServletResponse) {

       CustForm form = (CustForm) actionForm;
        try {
            String phone = form.getPhone();
            custserv.addCust(0, phone, "foo");
            return actionMapping.
                       findForward("success");
        } catch (ValidationException e) {
            return actionMapping.
                       findForward("invaliddata");
        }
    }
}
```

```
public interface CustomerService {
    void addCust(int i, String phone, String s)
            throws ValidationException;
}



public class CustomerServiceImpl
        implements CustomerService {

    public void addCust(int i,
                        String phone,
                        String s)
            throws ValidationException {

        PreparedStatement dbstmt = "INSERT …";

        if (!justnumbers(phone))
            throw new ValidationException();
        try {
            dbstmt.setInt(1, i);
            dbstmt.setString(3, phone);
            dbstmt.setString(4, s);
            dbstmt.executeUpdate();

        } catch (SQLException e) {
            throw new RuntimeException();
        }
    }
    static boolean justnumbers(String s) {
      return s.matches("[0-9]*");
    }
}
```

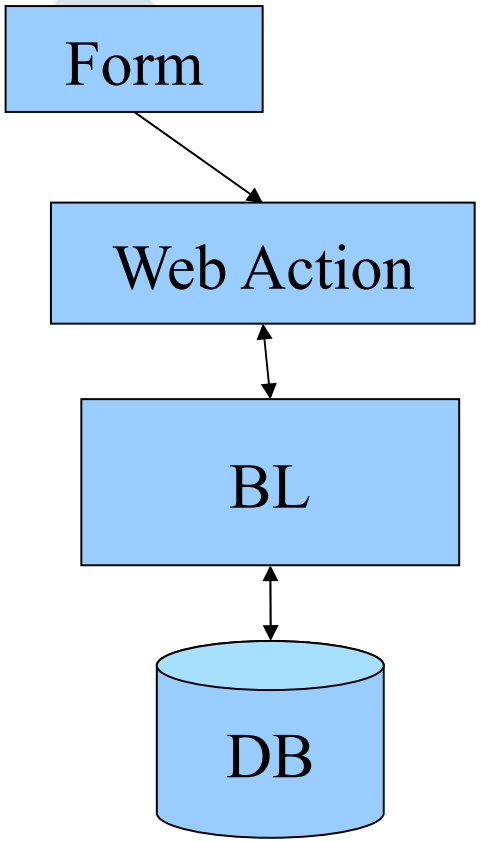
omega point.

```
public class CustForm extends ActionForm {
    String phone;
    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
}


public class AddCustAction extends Action {
    CustomerService custserv = null;

    @Override
    public ActionForward execute(
      ActionMapping actionMapping,
      ActionForm actionForm,
      HttpServletRequest httpServletRequest,
      HttpServletResponse httpServletResponse) {

       CustForm form = (CustForm) actionForm;
        try {
            String phone = form.getPhone();
            custserv.addCust(0, phone, "foo");
            return actionMapping.
                       findForward("success");
        } catch (ValidationException e) {
            return actionMapping.
                       findForward("invaliddata");
        }
    }
}
```

```
public interface CustomerService {
    void addCust(int i, String phone, String s)
            throws ValidationException;
}


public class CustomerServiceImpl
        implements CustomerService {

    public void addCust(int i,
                        String phone,
                        String s)
            throws ValidationException {

        PreparedStatement dbstmt = "INSERT …";

        if (!justnumbers(phone))
            throw new ValidationException();
        try {
            dbstmt.setInt(1, i);
            dbstmt.setString(3, phone);
            dbstmt.setString(4, s);
            dbstmt.executeUpdate();

        } catch (SQLException e) {
            throw new RuntimeException();
        }
    }
    static boolean justnumbers(String s) {
      return s.matches("[0-9]*");
    }
}
```

omega point.

Phone Number = Strings all the way




```
class CustForm extends ActionForm
   private String phone

class AddCustAction extends Action ... ...
   execute(...)
    String phone = form.getPhone();
    custserv.addCust(..., phone, ...);

class CustomerServiceBean ...
   void addCust(..., String phone, ...)
   throws ValidationException
        ...
        if (!justnumbers(phone)) ...
           throw new ValidationException();
        ...
        dbstmt.setString(4, phone);

   static boolean justnumbers(String s) ...
```

omega point.

Interpretations of Phone Number String

```
SalesRep findSalesRepresentative(String phone) {
  // phone directly assoc with sales rep?
  Object directrep = phone2repMap.get(phone);
  if (directrep != null)
    return (SalesRep) directrep;

  // find area code
  String prefix = null;
  for (int i=0; i<phone.length(); i++){
    String begin = phone.subString(0,i);
    if(isAreaCode(begin)) {
      prefix = begin;
      break;
    }
  }
  String areacode = prefix;

  // exists area representative?
  Object arearep = area2repMap.get(areacode);
  if (arearep != null)
    return (SalesRep) arearep;

  // neither direct nor area sales representative
  return null;
}
```

omega point.

Make Implicit Concepts Explicit

- Phone Number implicit
- Does it cause trouble?
  - Bugs
  - Awkward code
  - Duplication

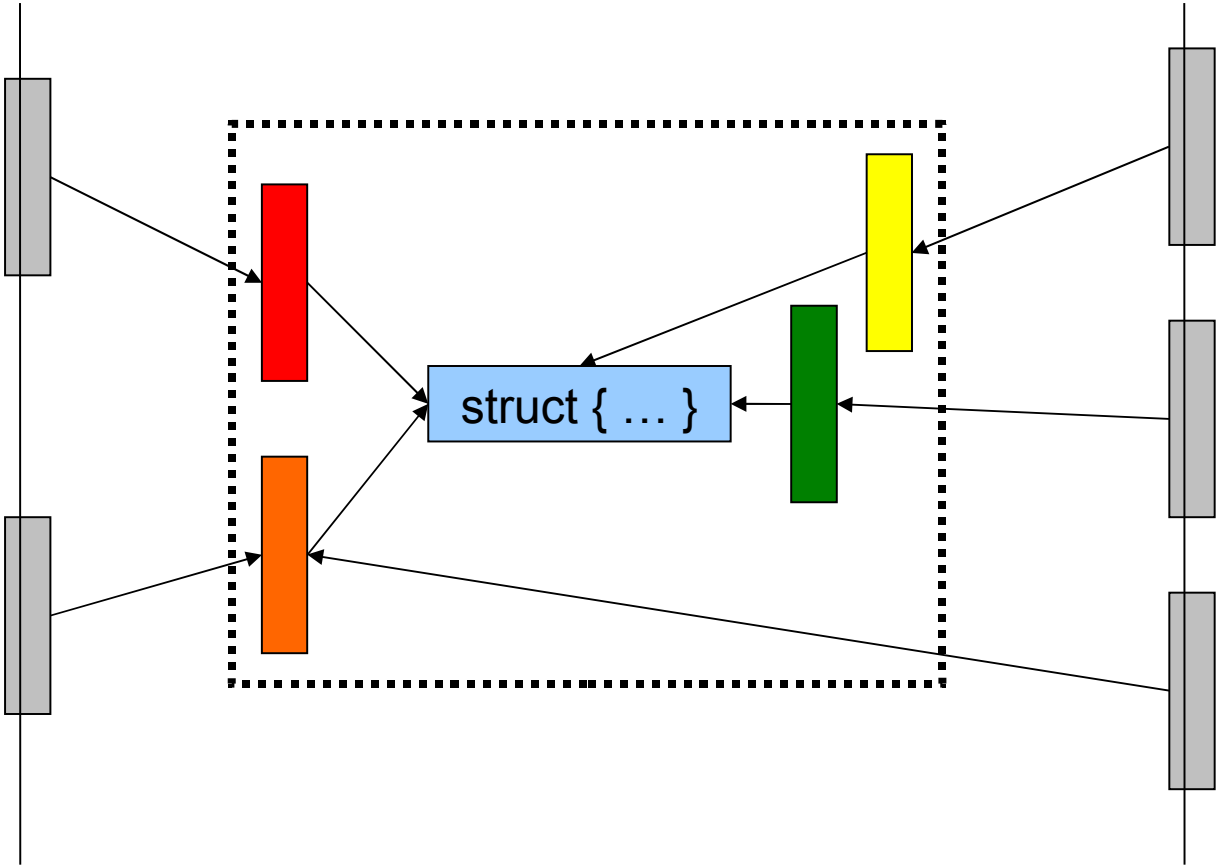
- Enrich language with new concept
  - Glossary
  - Code

Enter: Domain Logical Value Object, String Wrap Style

```
public class PhoneNumber {
    private final String number;

    public PhoneNumber(String number) {
     if(!isValid(number))
             throw …
             this.number = number;
    }

    public String getNumber() {
       return number;
      }

    static public boolean isValid(String number) {
       return number.matches("[0-9]*");
    }

    public String getAreaCode() {
      String prefix = null;
      for (int i=0; i< number.length(); i++){
        String begin = number.subString(0,i);
        if(isAreaCode(begin)) {
          prefix = begin;
          break;
      }
      return prefix;
    }

    private boolean isAreaCode(String prefix) { ... }
}
```

omega point.

Data as Centres of Gravity



omega point.

Evaluation Time

Did it get any better?

How about:

- Service API clarity?
- In-Data Validation and Error Handling?
- Clarity of Business Tier Code?
- Testability?

omega point.

Service API Clarity

```
void addCust(String,
   String, String,
   int,
   int,
   String, String,
   boolean)
```

Can you clarify that, please?

omega point.

Service API Clarity

```
void addCust(Name,
   PhoneNumber, PhoneNumber,
   CreditStatus,
   SalesRepId,
   Name, PhoneNumber,
   ParnerStatus)
```

omega point.

In-Data Validation and Error Handling

```
class CustForm extends ActionForm
   private String phone
   ...


class AddCustAction extends Action
   ... execute(...)
    custserv.addCust(...,
        form.getPhone(), ...)

class CustomerServiceBean ...
   void addCust(..., String phone,...)
   throws ValidationException
       ...
       if (!justnumbers(phone))
          throw new ValidationException();
```

omega point.

In-Data Validation and Error Handling

```
class CustForm extends ActionForm
   private String phone
   ... validate() ...
       if(!PhoneNumber.isValid(phone))
       ...

class AddCustAction extends Action
   ... execute(...)
       custserv.addCust(...,
              new PhoneNumber(form.getPhone()), ...)

class CustomerServiceBean ...
   void addCust(..., PhoneNumber phone,...)
   throws ValidationException
       ...
       if (!justnumbers(phone))
          throw new ValidationException();
```

omega point.

Focus of Business Logic Tier Code

```
SalesRep findSalesRepresentative(String phone) {
  // phone directly assoc with sales rep?
  Object directrep = phone2repMap.get(phone);
  if (directrep != null)
    return (SalesRep) directrep;

  // find area code
  String prefix = null;
  for (int i=0; i<phone.length(); i++){
    String begin = phone.subString(0,i);
    if(isAreaCode(begin)) {
      prefix = begin;
      break;
    }
  }
  String areacode = prefix;

  // exists area representative?
  Object arearep = area2repMap.get(areacode);
  if (arearep != null)
    return (SalesRep) arearep;

  // neither direct nor area sales representative
  return null;
}
```

omega point.

Focus of Business Logic Tier Code

```
SalesRep findSalesRepresentative(PhoneNumber phone) {
  // phone directly assoc with sales rep?
  Object directrep = phone2repMap.get(phone);
  if (directrep != null)
    return (SalesRep) directrep;



   // junk deleted









  // exists area representative?
  Object arearep = area2repMap.get(phone.getAreaCode());
  if (arearep != null)
    return (SalesRep) arearep;

  // neither direct nor area sales representative
  return null;
}
```

omega point.

Testability: Test code CustomerService erroneous phone

```
public void testShouldDetectNullPhone() {
    try {
        String phone = null;
        out.addCust("name", phone, null, 0, 0, "", null, false);
        fail();
    } catch (NullPointerException e) { /*ok*/ }
}

public void testShouldDetectInvalidPhone() {
    try {
        String phone = "not a phone number";
        out.addCust("name", phone, null, 0, 0, "", null, false);
        fail();
    } catch (ValidationException e) { /*ok*/ }
}

public void testShouldDetectEmptyPhone() {
    try {
        String phone = "";
        out.addCust("name", phone, null, 0, 0, "", null, false);
        fail();
    } catch (ValidationException e) { /*ok*/ }
}

public void testShouldDetectPhoneWithPlusInTheMiddle() {
    try {
        String phone = "46+709158843";
        out.addCust("name", phone, null, 0, 0, "", null, false);
        fail();
    } catch (ValidationException e) { /*ok*/ }
}
```

omega point.

Testability: Test code CustomerService erroneous fax

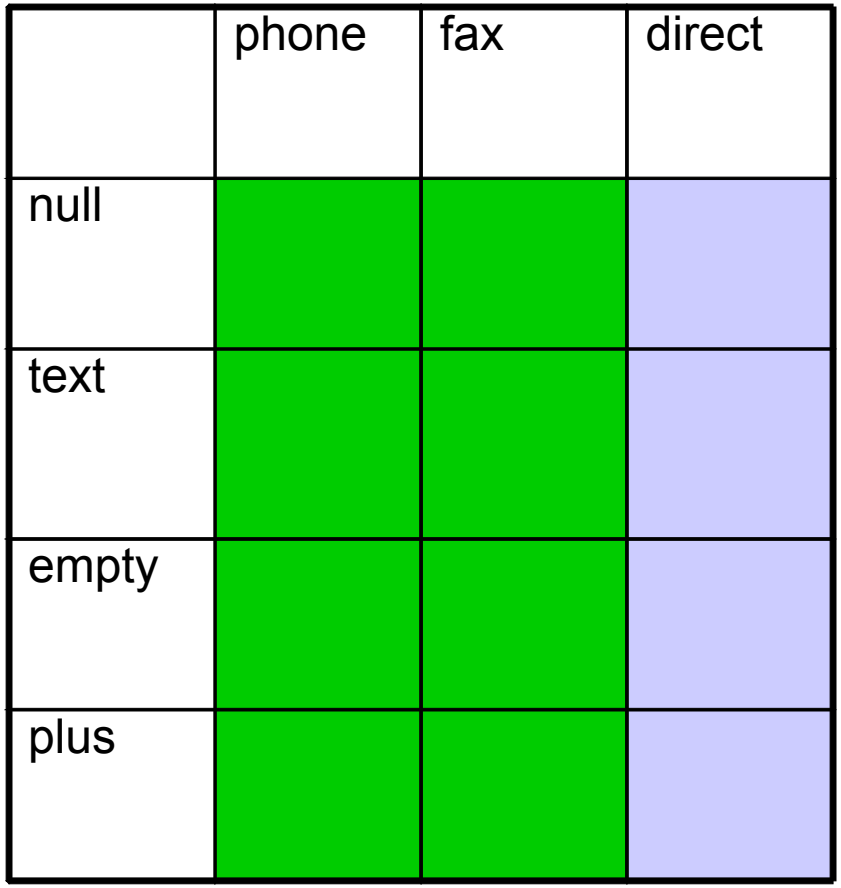
```
public void testShouldDetectNullFax() {
    try {
        String fax = null;
        out.addCust("name", "40068", fax, 0, 0, "", null, false);
        fail();
    } catch (NullPointerException e) { /*ok*/ }
}

public void testShouldDetectInvalidFax() {
    try {
        String fax = "not a phone number";
        out.addCust("name", "40068", fax, 0, 0, "", null, false);
        fail();
    } catch (ValidationException e) { /*ok*/ }
}

public void testShouldDetectEmptyFax() {
    try {
        String fax = "";
        out.addCust("name", "40068", fax, 0, 0, "", null, false);
        fail();
    } catch (ValidationException e) { /*ok*/ }
}

public void testShouldDetectFaxWithPlusInTheMiddle() {
    try {
        String fax= "46+709158843";
        out.addCust("name", "40068", fax, 0, 0, "", null, false);
        fail();
    } catch (ValidationException e) { /*ok*/ }
}
```

omega point.

Number of tests

- 4 cases
- 3 uses

Total = m * n =
12 tests



omega point.

Testability: Test code - PhoneNumber

```
public void testShouldNotAcceptNullNumber()
    try {
      new PhoneNumber(null);
      fail();
    } catch (NullPointerException e) { /*ok*/ }
  }

public void testShouldConsiderEmptyNumberAsInvalid()
    try {
      new PhoneNumber("");
      fail();
    } catch (IllegalArgumentException e) { /*ok*/ }
  }

public void testShouldConsiderRandomTextAsInvalid()
    try {
      new PhoneNumber("This is not a phone number");
      fail();
    } catch (IllegalArgumentException e) { /*ok*/ }
  }

public void testShouldConsiderPlusInMiddleAsInvalid()
    try {
      new PhoneNumber("46+709158843");
      fail();
    } catch (IllegalArgumentException e) { /*ok*/ }
  }
```

omega point.

Testability: Test code – CustomerService

```
static private VALID_PHONE = new PhoneNumber("40068”)

public void testShouldDetectNullPhone() {
        try {
            PhoneNumber phone = null;
            out.addCust("name", phone, VALID_PHONE, 0, 0, "", VALID_PHONE, false);
            fail();
        } catch (NullPointerException e) { /*ok*/ }
    }

public void testShouldDetectNullFax() {
        try {
            PhoneNumber fax = null;
            out.addCust("name", VALID_PHONE, fax, 0, 0, "", VALID_PHONE, false);
            fail();
        } catch (NullPointerException e) { /*ok*/ }
    }

public void testShouldDetectNullDirectNumber() {
        try {
            PhoneNumber direcr = null;
            out.addCust("name", VALID_PHONE, VALID_PHONE, 0, 0, "", direct, false);
            fail();
        } catch (NullPointerException e) { /*ok*/ }
    }
```

omega point.

Number of tests

- 4 cases
- 3 uses

Total = m + n =
7 tests

	Phone Number	phone	fax	direct
null				
text				
empty				
plus				

omega point.

Evaluation Summary

API	ambiguous	readable
validation error handling	all over and deep down	pushed to border
clarity of business code	detail clutter	lucent
testability	m*m	m+n

omega point.

Bonus!

Note:

No changes to

- Directory hierarchy
- Deployment routines
- Build scripts
- Classpath
- etc

“Monday morning compliant!”

omega point.

Candidates for DLVO

- Strings with format limitations
  - Name
  - Ordernumber
  - Zipcode
- Integers with limitations
  - Percentage (0-100%)
  - Quantity (≥ 0)
- Arguments/return values in service methods
  - Double
  - Map<String, List<Integers>>

omega point.

Side note: COP/Qi4J

- Composite Oriented Programming (Rickard Öberg)
- Qi4J: COP framework on Java (www.qi4j.org)
- Using DDD terminology / DDD enabling
- ValueComposite
  - @Immutable
  - Equals is defined by the values
  - Properties
    - discrete type
    - serializable object
    - ValueComposite

omega point.

1 minute break

omega point.

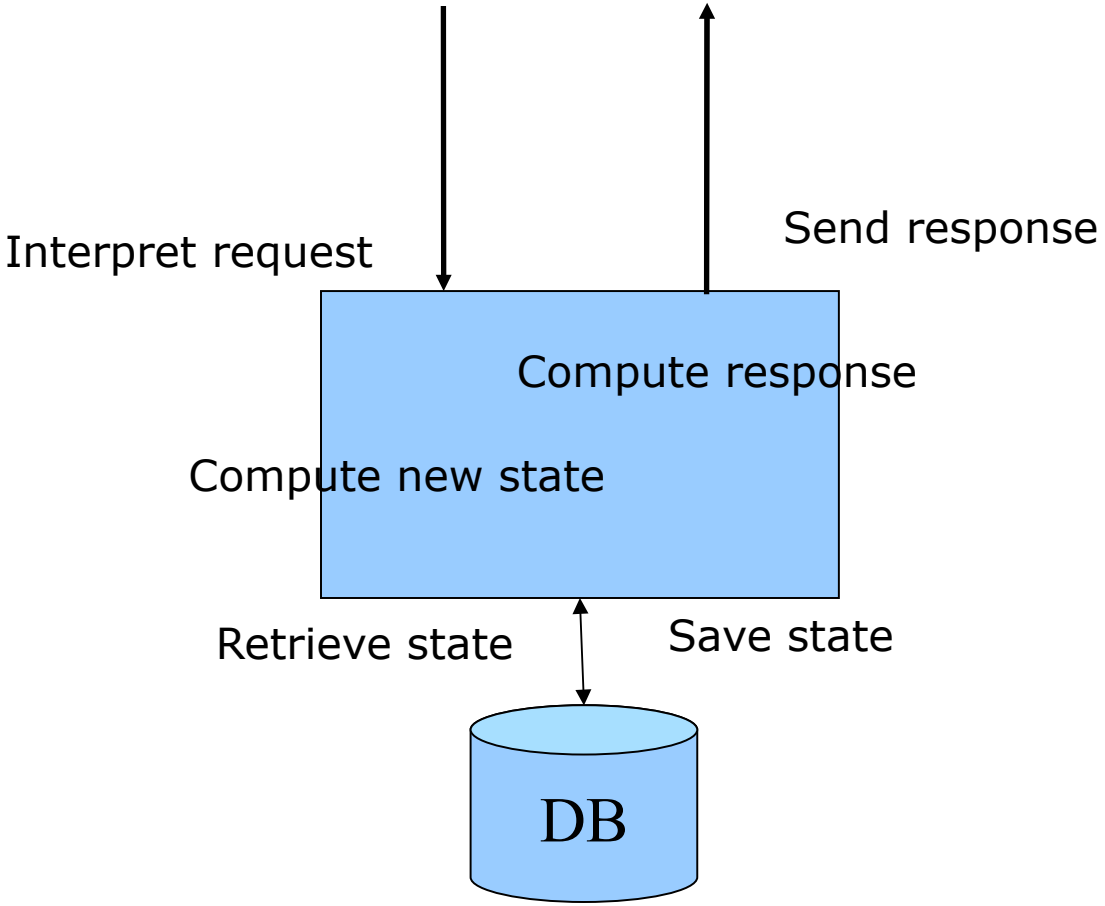
Warmed Up – Ready for Take-Off

”I’d like to leave you a little bit confused …
because confusion is creative”

- Swiss dance instructor,
now living in San Francisco

omega point.

Things done on server

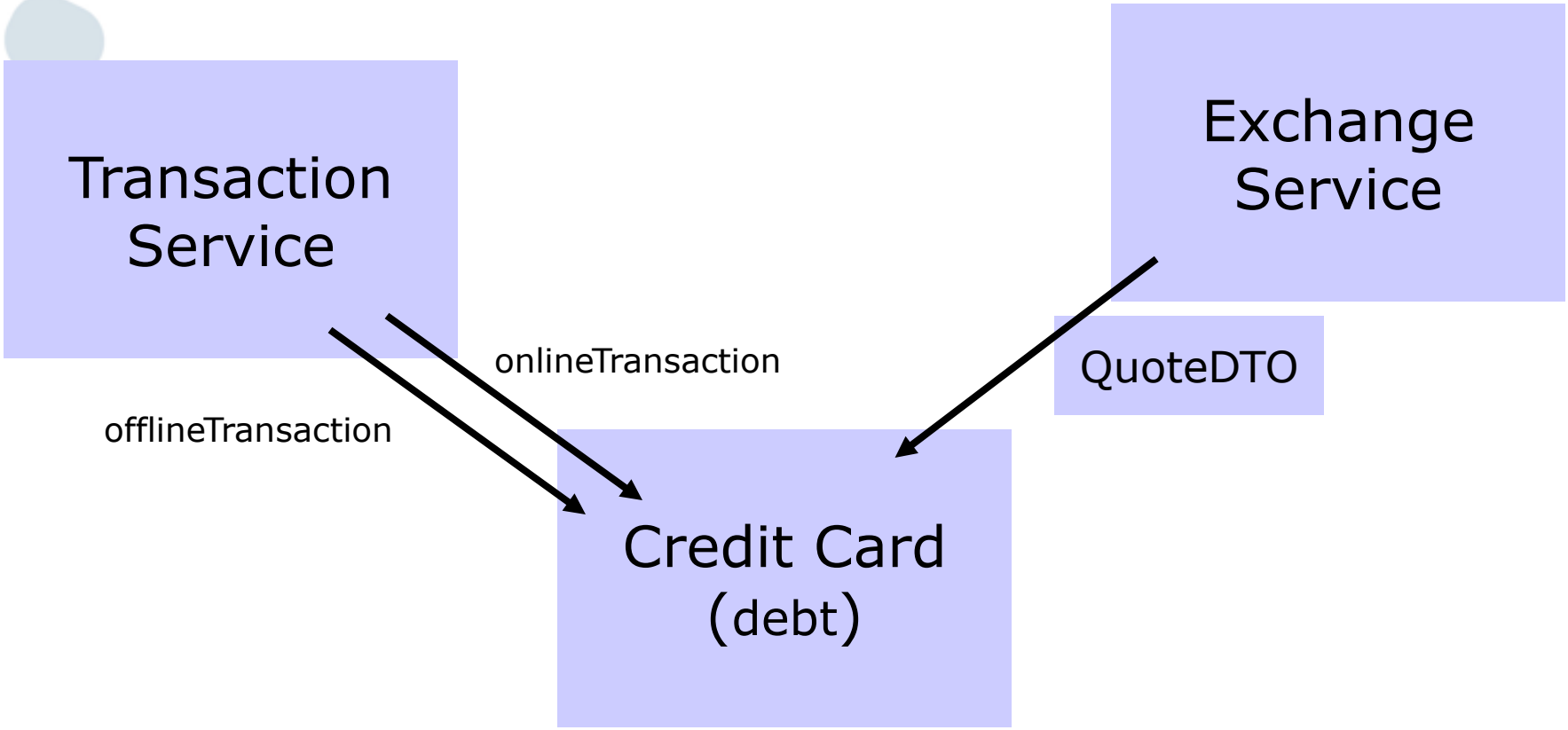


omega point.

Credit Card and Multiple Currencies

omega point.

Architecture



omega point.

Credit Card Entity

```
public interface CardRegistry {
    CreditCard find(CardNumber number);
}

public class CreditCard {
   CardNumber number;
   Currency cardcurrency;
   BigDecimal debt;
   void onlineTransaction(StoreId store, BigDecimal amount)
   void offlineTransaction(StoreId store, BigDecimal amount,
    Date transactionDay)
}
```

omega point.

Exchange Service

```
public interface ExchangeService {
  Currency REF_CURR = Currency.getInstance("EUR");
  BigDecimal FEE = BigDecimal.ONE;
  List<QuoteDTO> findRate(Currency currency);
  QuoteDTO findCurrentRate(Currency currency);
}

public class QuoteDTO {
  Currency currency;
  BigDecimal rate; // relative reference currency
  Date validfromday;
  Date validtoday;
}
```

omega point.

onlineTransaction(...)

```
void onlineTransaction(StoreId store, BigDecimal amount) {
        Currency storeCurrency = storeService.getCurrency(store);
        if (storeCurrency.equals(this.cardcurrency)) {
            debt = debt.add(amount);
        } else if (cardcurrency.equals(ExchangeService.REF_CURR) &&
                (!storeCurrency.equals(ExchangeService.REF_CURR))){
            QuoteDTO storequote =
          exchange.findCurrentRate(storeCurrency);
            debt = debt.add(amount.multiply(storequote.rate))
          .add(ExchangeService.FEE);
        } else if (!cardcurrency.equals(ExchangeService.REF_CURR) &&
                (storeCurrency.equals(ExchangeService.REF_CURR))){
            QuoteDTO cardquote = exchange.findCurrentRate(cardcurrency);
            debt = debt.add(amount.divide(cardquote.rate))
          .add(ExchangeService.FEE);
        } else {
            QuoteDTO cardquote = exchange.findCurrentRate(cardcurrency);
            QuoteDTO storequote = exchange.findCurrentRate(storeCurrency);
            debt = debt.add(amount.divide(cardquote.rate)
               .multiply(storequote.rate))
                    .add(ExchangeService.FEE.multiply(BigDecimal.valueOf(2)));
        }
    }
}
```

omega point.

onlineTransaction(...)

```
void onlineTransaction(StoreId store, BigDecimal amount) {
        Currency storeCurrency = storeService.getCurrency(store);
        if (storeCurrency.equals(this.cardcurrency)) {
            debt = debt.add(amount);
        } else if (cardcurrency.equals(ExchangeService.REF_CURR) &&
                (!storeCurrency.equals(ExchangeService.REF_CURR))){
            QuoteDTO storequote =
          exchange.findCurrentRate(storeCurrency);
            debt = debt.add(amount.multiply(storequote.rate))
          .add(ExchangeService.FEE);
        } else if (!cardcurrency.equals(ExchangeService.REF_CURR) &&
                (storeCurrency.equals(ExchangeService.REF_CURR))){
            QuoteDTO cardquote = exchange.findCurrentRate(cardcurrency);
            debt = debt.add(amount.divide(cardquote.rate))
          .add(ExchangeService.FEE);
        } else {
            QuoteDTO cardquote = exchange.findCurrentRate(cardcurrency);
            QuoteDTO storequote = exchange.findCurrentRate(storeCurrency);
            debt = debt.add(amount.divide(cardquote.rate)
               .multiply(storequote.rate))
                    .add(ExchangeService.FEE.multiply(BigDecimal.valueOf(2)));
        }
    }
```

omega point.

offlineTransaction(...)

```
void offlineTransaction(StoreId store, Amount amount,
    Date purchaseday) {
    Currency storeCurrency = storeService.getCurrency(store);
     List<QuoteDTO> quotes = exchange.findRate(storeCurrency);
     QuoteDTO found = null;
     for (QuoteDTO quote : quotes) {
     if (quote.validfrom.before(purchaseday)
         && quote.validto.after(purchaseday)) {
              found = quote;
              break;
          }
     }
     if (found == null)
          throw new RateException("rate not found");
     // ... and for card currency
     // ... and convert
     // ... add increase debt
}
```

omega point.

Problem

Entity burdened with details

- Keeping track of currencies
- Performing exchange
- Quote validity

Burn-Out!

Compound value objects enter stage

Buckle up

omega point.

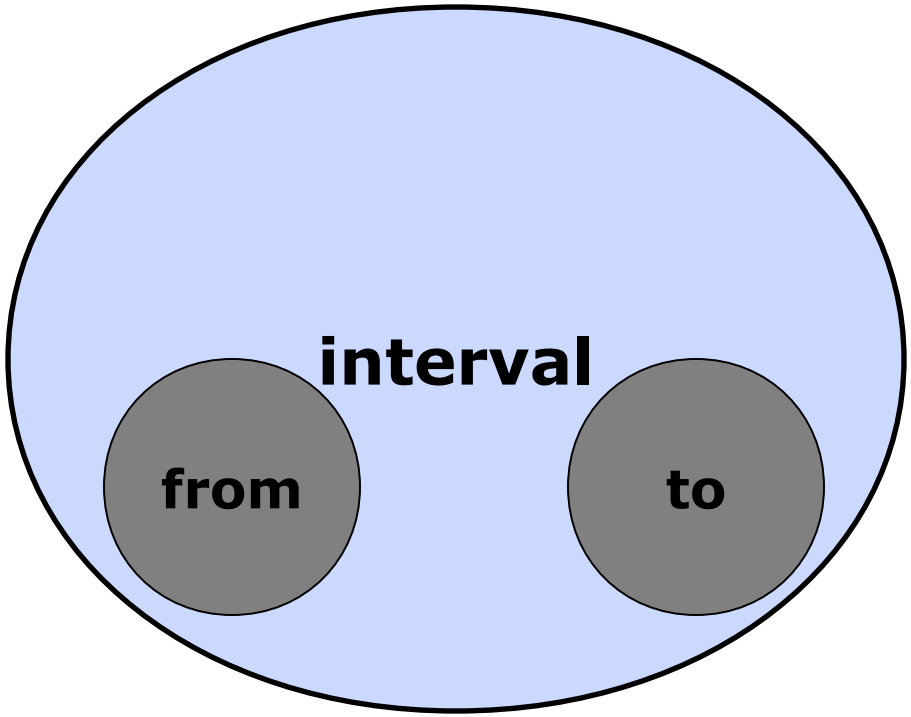
Tricks to Use

- Encapsulate Multi-Object Behaviour
- Make Implicit Context Explicit
  - Encapsulate Context

Refactoring: Encapsulate multi-object behaviour

- validfrom, validto, validity

```
if (quote.validfrom.before(purchaseday)
    && quote.validto.after(purchaseday)) {
```



omega point.

Refactoring: Introduce Data Pair Object

- Date + Date = TimeInterval

```
if (quote.validinterval.contains(purchaseday)) {

class TimeInterval {
   Date from;
   Date to;
   boolean contains(Date day) ...

class QuoteDTO {
   Currency currency;
   BigDecimal rate;
   TimeInterval validinterval;
```

- Why not use QuoteDTO?

omega point.

DTOs and VOs

- DTO – Data Transfer Object
  - purpose: data transfer – technical construct
  - bunch of data – not necessarily coherent
  - no/little behaviour
- VO – Value Object
  - purpose: domain representation
  - high-coherent data
  - rich on behaviour

omega point.

Implicit Context Problem

- Amount of what?

```
void offlineTransaction(StoreId store,
   BigDecimal amount,
   Date purchaseday) {

   Currency storeCurrency = storeService.getCurrency(store);
   ...
   debt.add(amount);

public class CreditCard {
   CardNumber number;
   Currency cardcurrency;
   BigDecimal debt;
```

- Context knowledge in caller / surrounding

```
void debitCustomer( … ) {
   CreditCard card = cardReg.find(cardNumber);
   card.offlineTransaction(store, amount, date);
```

omega point.

Make Context Explicit

- BigDecimal + Currency = Money

```
void offlineTransaction(Money money, Date purchaseday) {

public class CreditCard {
   CardNumber number;
   Money debt;

public class Money {
  Money add(Money money) ... // check same currency

void debitCustomer( … ) {
   CreditCard card = cardReg.find(cardNumber);
   Currency storeCurrency = storeService.getCurrency(store);
   Money money = new Money(amount, storeCurrency);
   card.offlineTransaction(money, date);
```

omega point.

ExchangeService/QuoteDTO: Implicit Context in Service Design

```
public interface ExchangeService {
  Currency REF_CURR = Currency.getInstance("EUR");
  BigDecimal FEE = BigDecimal.ONE;
  List<QuoteDTO> findRate(Currency currency);
  QuoteDTO findCurrentRate(Currency currency);
}

public class QuoteDTO {
  Currency currency;
  BigDecimal rate; // relative reference currency
  TimeInterval validinterval;
}
```

omega point.

Service with Explicit Context

```
public interface ExchangeService {
  Currency REF_CURR = Currency.getInstance("EUR");
  BigDecimal FEE = BigDecimal.ONE;
  List<QuoteDTO> findRate(Currency currency);
  QuoteDTO findCurrentRate(Currency from, Currency to);
}

public class QuoteDTO {
    Currency from; // made explicit
    Currency to;
    BigDecimal rate;
    TimeInterval validinterval;
}
```

omega point.

Compound Coherent Data – Encapsulate Multi-Object Behaviour

from, to, rate – exchange logic

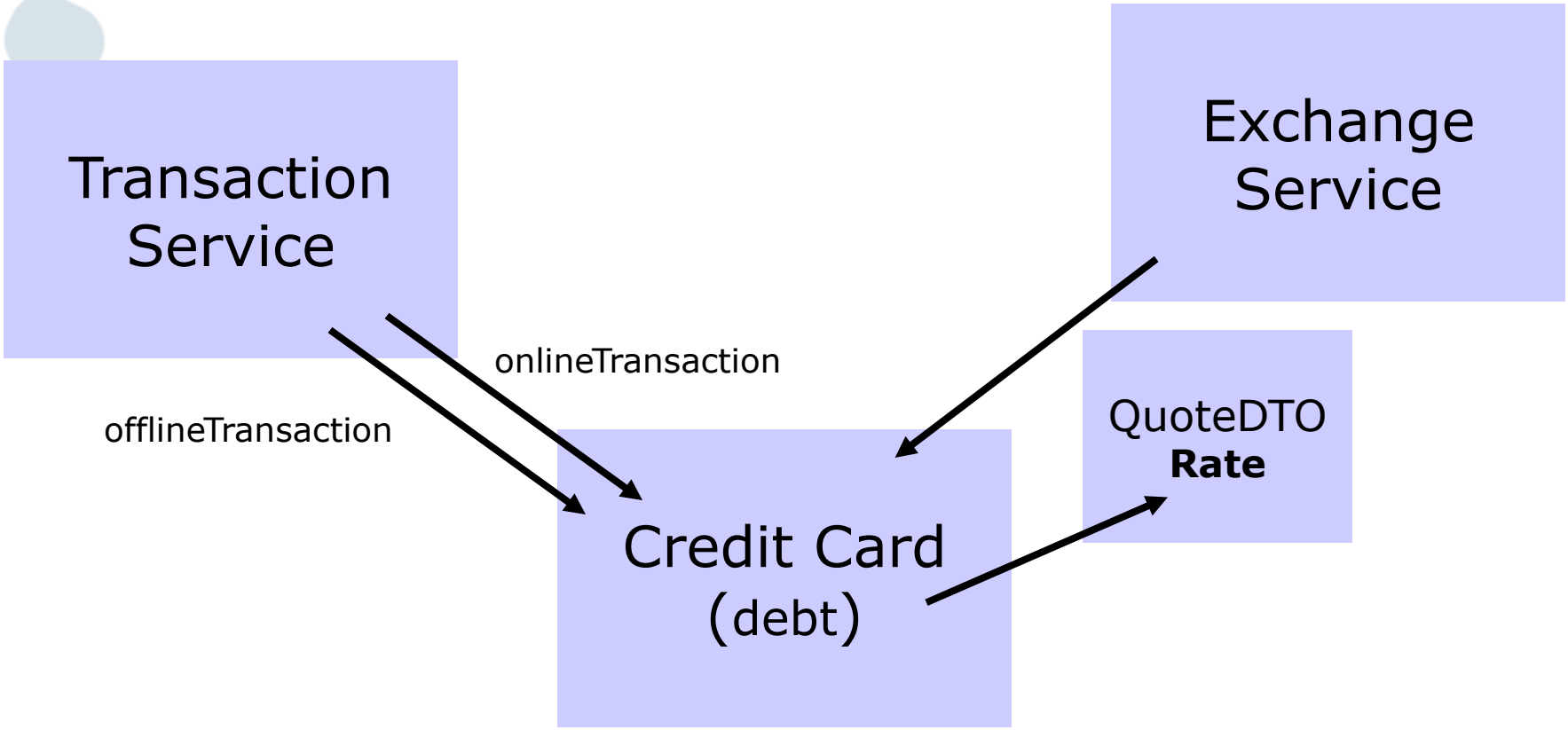
```
debt.add(amount.divide(cardquote.rate).multiply(storequote.rate))
```

Currency + Currency + BigDecimal = Rate

```
class Rate {
    Currency from;
    Currency to;
    BigDecimal rate;
    Money exchange(Money m) {
     if(!m.currency.equals(from)) throw new ...
     return new Money(m.amount.divide(rate),to);

public class QuoteDTO {
    Rate rate;
    TimeInterval validinterval;
}

class CreditCard {
    void onlineTransaction(Money money) {
     debt = debt.add(rate.exchage(money));
```

omega point.

Architecture



omega point.

ExchangeService with smart Rates

ExchangeService returning DTO

- Just data
- Computations performed by client
- Cannot extend behaviour

ExchangeService returning Rates (VO)

- Object with encapuslated behaviour
- Computations performed by returned object
- Can extend value objects with more behaviour

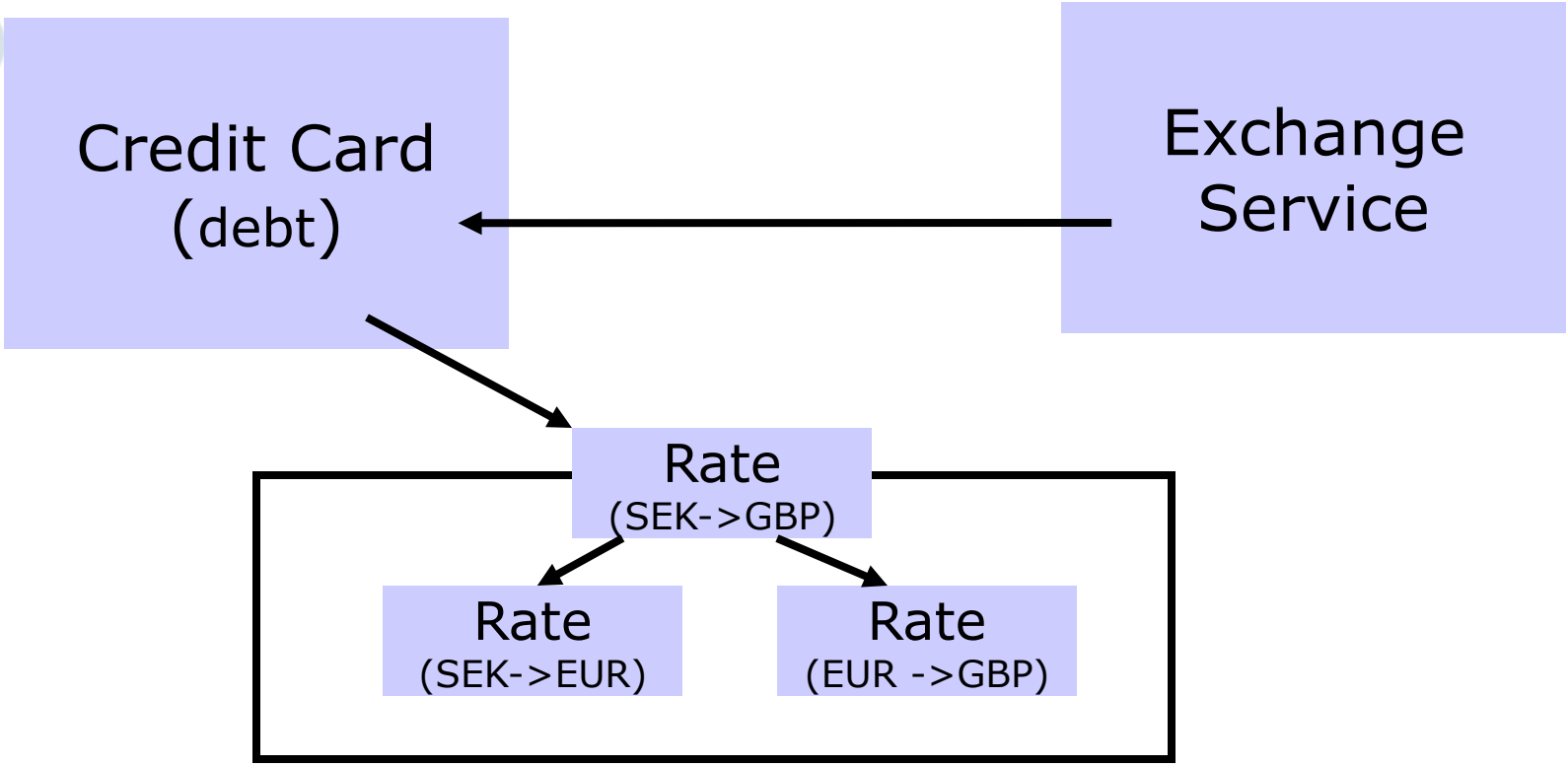
omega point.

What about Fees?

- Always ask ExchangeService for quote with rate
- Rate is intelligent object
  - exchange method can calculate fee
- Rate GBP->GBP will have no fee
- Rate GBP -> REF_CURR will have fee
- Rate REF_CURR -> SEK will have fee

omega point.

... and Two-Step Exchange?



omega point.

Composite Rate

- SEK -> GBP
  - SEK -> REF_CURR
  - REF_CURR -> GBP

```
class CompositeRate extends /*implements*/ Rate {
  Rate first;
  Rate second;
  Money exchange(Money amt) {
   return second.exchange(first.exchange(amt));
  }
}
```

omega point.

Finally DTO -> VO

```
public class Quote {
  Rate rate;
  TimeInterval validinterval;
  Quote(Currency from, Currency to,
          BigDecimal fromrate, BigDecimal torate,
          Date validfrom, Date validto) {
    rate = new CompositRate(
                new SimpleRate(from, REF_CURR, fromrate),
                new SimpleRate(REF_CURR, to, torate));
    validinterval = new TimeInterval(validfrom, validto);
  Money exchange(Money m, Date day) {
    if(!validinterval.contain(day)) throw new ...
    return rate.exchange(m);
  }
}
```

omega point.

Where did this Take Us?

- Context-Aware Client Code
- Smart Exchange Service
- Library with API
- Some Things Left in Entity

omega point.

Context-Aware Client

```
class TransactionService
   void debitCustomer( … ) {
        CreditCard card = cardReg.find(cardNumber);
        Currency storeCurr = storeService.getCurrency(store);
        Money money = new Money(amount, storeCurr);
   card.onlineTransaction(money);
        ...
   }
}
```

omega point.

Smart Service

```
class ExchangeServiceImpl {
  Quote findCurrentRate(Currency from,
                        Currency to) {
    ... // db SELECT
    return new Quote ...
      // new CompositeRate( ... );
```

omega point.

Library with API

```
class TimeInterval {
   boolean contains(Date day)

public class Money {
   Money add(Money money)
   ... // check same currency

interface Rate {
   Money exchange(Money m) ...

class SimpleRate implements
   Rate{
   Currency from;
   Currency to;
   BigDecimal rate;
   Money exchange(Money m)
   ...  // including fee
```

```
class CompositeRate implements
   Rate {
   Rate first;
   Rate second;
   Money exchange(Money m)
   ... // two step exchange

public class Quote {
   Rate rate;
   TimeInterval validinterval;
   Money exchange(Money m,
        Date day)
```

omega point.

What about the Value Object Library?

- How can we test?
- What can we test?
- Concurrency issues?
- Possible to audit code?

omega point.

What is Left in Card Entity?

omega point.

onlineTransaction(...)

```
void onlineTransaction(StoreId store, BigDecimal amount) {
        Currency storeCurrency = storeService.getCurrency(store);
        if (storeCurrency.equals(cardcurrency)) {
            debt = debt.add(amount);
        } else if (cardcurrency.equals(ExchangeService.REF_CURR) &&
                (!storeCurrency.equals(ExchangeService.REF_CURR))){
            QuoteDTO storequote =
          exchange.findCurrentRate(storeCurrency);
            debt = debt.add(amount.multiply(storequote.rate))
          .add(ExchangeService.FEE);
        } else if (!cardcurrency.equals(ExchangeService.REF_CURR) &&
                (storeCurrency.equals(ExchangeService.REF_CURR))){
            QuoteDTO cardquote = exchange.findCurrentRate(cardcurrency);
            debt = debt.add(amount.divide(cardquote.rate))
          .add(ExchangeService.FEE);
        } else {
            QuoteDTO cardquote = exchange.findCurrentRate(cardcurrency);
            QuoteDTO storequote = exchange.findCurrentRate(storeCurrency);
            debt = debt.add(amount.divide(cardquote.rate)
               .multiply(storequote.rate))
                    .add(ExchangeService.FEE.multiply(BigDecimal.valueOf(2)));
        }
    }
```

omega point.

onlineTransaction(...)

```
void onlineTransaction(Money m){
   Quote quote =
     exchange.findCurrentRate(m.getCurrency(),
   cardcurrency);
   debt = debt.add(quote.exchange(m, new Date());
}
```

- Yes, currency check included
- Yes, validity check included
- Yes, exchange computation included
- Yes, fees included

omega point.

Tricks Used

- Simple Value Objects
- Data as Centres of Gravity
- Encapsulate Multi-Object Behaviour
- Make Context Explicit

Analysis

- Computation complexity moved to value objects
  - Adding new terminology to language
- Compound value objects can swallow lots of computational complexity
  - Provides advanced language
- Entity relieved of complexity
  - Uses advanced language
- Improved extensibility
  - esp clarity, testability and concurrency issues

omega point.

Overall Presentation Goal

Show how some power use of Value Objects can radically change design and code,

hopefully to the better

omega point.

Anything worth remembering?

- Three points to remember tomorrow
-
-
-

omega point.

</Power of Value>

Thanks for your attention

afterthoughts:

- dan.bergh.johnsson@omegapoint.se
- dearjunior.blogspot.com
- www.omegapoint.se

omega point.