Power of Value

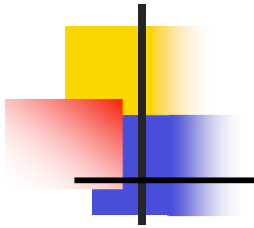
Domain Driven Design and Value Objects

Dan Bergh Johnsson
Partner and Spokesperson
Omegapoint AB, Sweden

```
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 true;
    }
}
```

```
 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)));
        }
    }
}
```

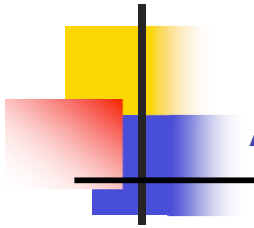


Overall Presentation Goal

Show how Value Objects can improve design and code clarity,

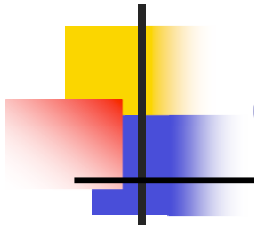
how to get started with them,

and some power usage



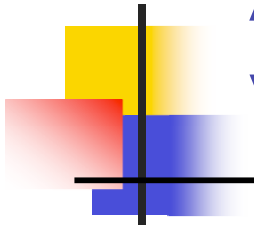
Analysis / Conclusions

- Computation complexity moved to value objects
- Compound value objects can swallow lots of computational complexity
- Entity (and Services) relieved of complexity
- Entity/Service code clearer



Get Value out of your Objects

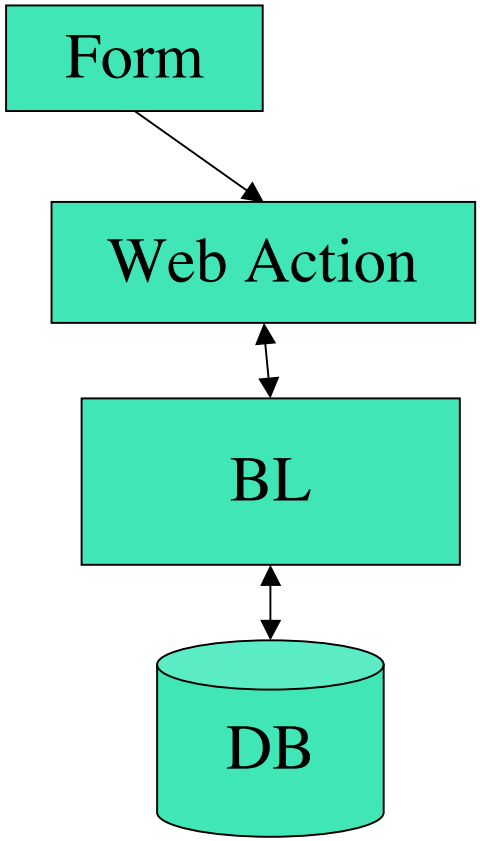
Get value out of your objects by getting Values out of your (Service and Entity) Objects.



Agenda: Value Objects, Domain Style

- Getting started
- Power use – extreme architectural makeover

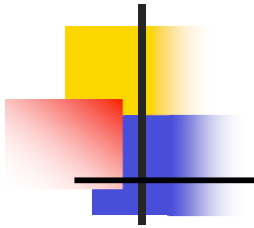
CRM Architecture Overview




```
class CustForm extends
  ActionForm

class AddCustAction extends
  Action ...
  ... execute(...)

class CustomerServiceBean ...
  void addCust(...)
```

```
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 = null/*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 true;
    }
}
```



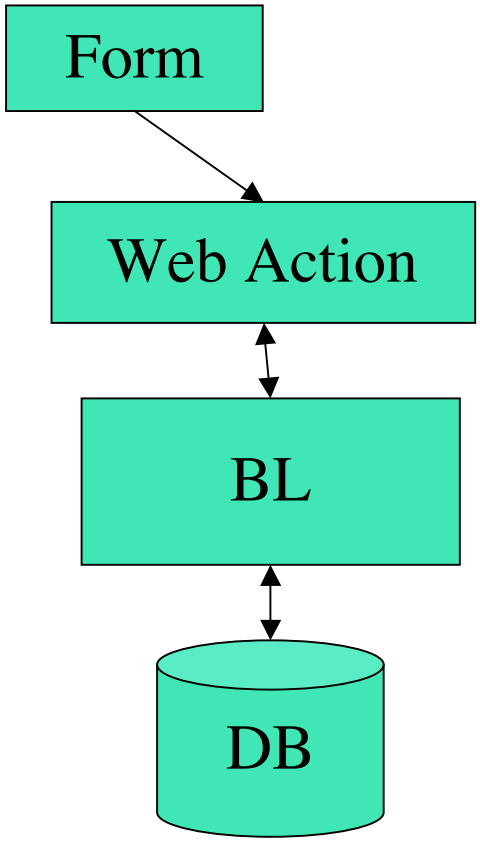
Method in Short

Relieve “component objects” from design/code complexity of basic format control et al.

Create simple value objects to host basic complexity.

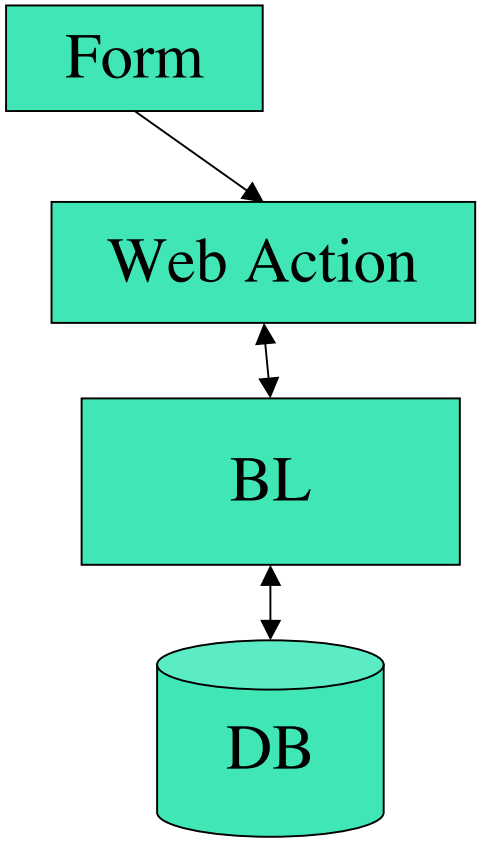
Look at the domain to find suitable candidates for value objects.

Adding Customer with Phone Number

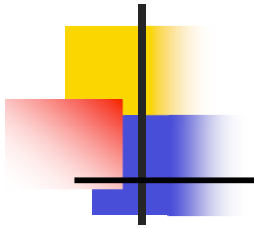



```
class CustForm extends ActionForm
   private String phone

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) ...
```

Phone Number = Strings all the way



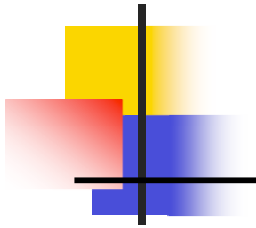

```
class CustForm extends ActionForm
   private String phone

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) ...
```



Implicit Important Concept

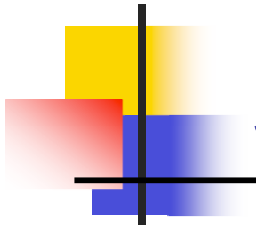
- Phone Number implicit
- Seems central enough

- Fair enough, as long as it does not cause trouble
  - Bugs
  - Awkward code
  - Duplication



Problems

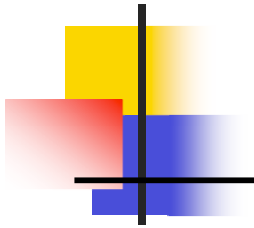
- Service API clarity
- In-Data Validation and Error Handling
- Clarity of Business Tier Code
- Testability



Service API Clarity

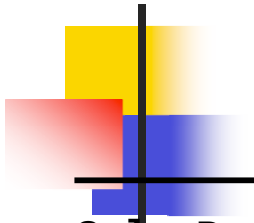
void addCust(String , *String*, String, int, int, String, String, boolean)

Can you clarify that, please?



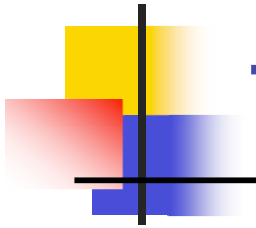
In-Data Validation and Error Handling

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



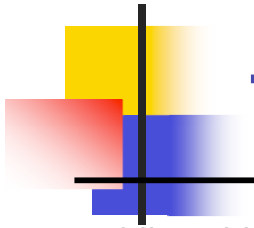
Focus of Business Logic Tier Code

```
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;
}
```



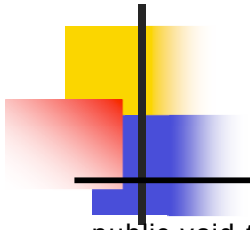
Testability

- Heavy testing
  - Need to test CustomerService to test phone numbers
  - Craves deployment of component
  - Probably slow
- Does the format control always work?
  - Different cases
  - Different uses
  - Hard to make extensive



Test code – 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*/ }
}
```



Test code – erroneous fax

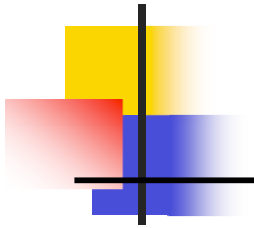
```
public void testShouldDetectNullFax() {
    try {
        String fax = null;
        out.addCust("name", "40068", fax, 0, 0, "", null, false);
        fail();
    } catch (NullPointerException e) { /*ok*/ }
}

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*/ }
}
```

Number of tests

- 4 cases
- 3 uses

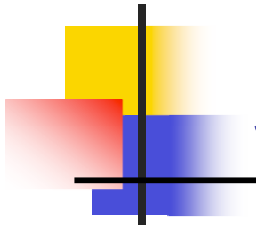
Total = m * n =
12 tests

	phone	fax	direct
null			
text			
empty			
plus			



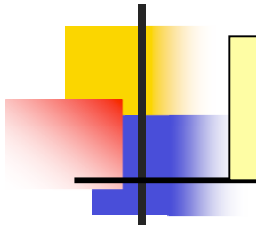
Problems

- Service API clarity
  - not clear
- In-Data Validation and Error Handling
  - in the wrong place
- Clarity of Business Tier Code
  - cluttered
- Testability
  - Detestably slow and cumbersome



Solution DDD-style

- Domain Driven Design
- *If* domain understanding is *critical*, then focus on *modelling* the domain
- “Take OO seriously”
- Use domain as guiding principle



Make Implicit Concepts Explicit

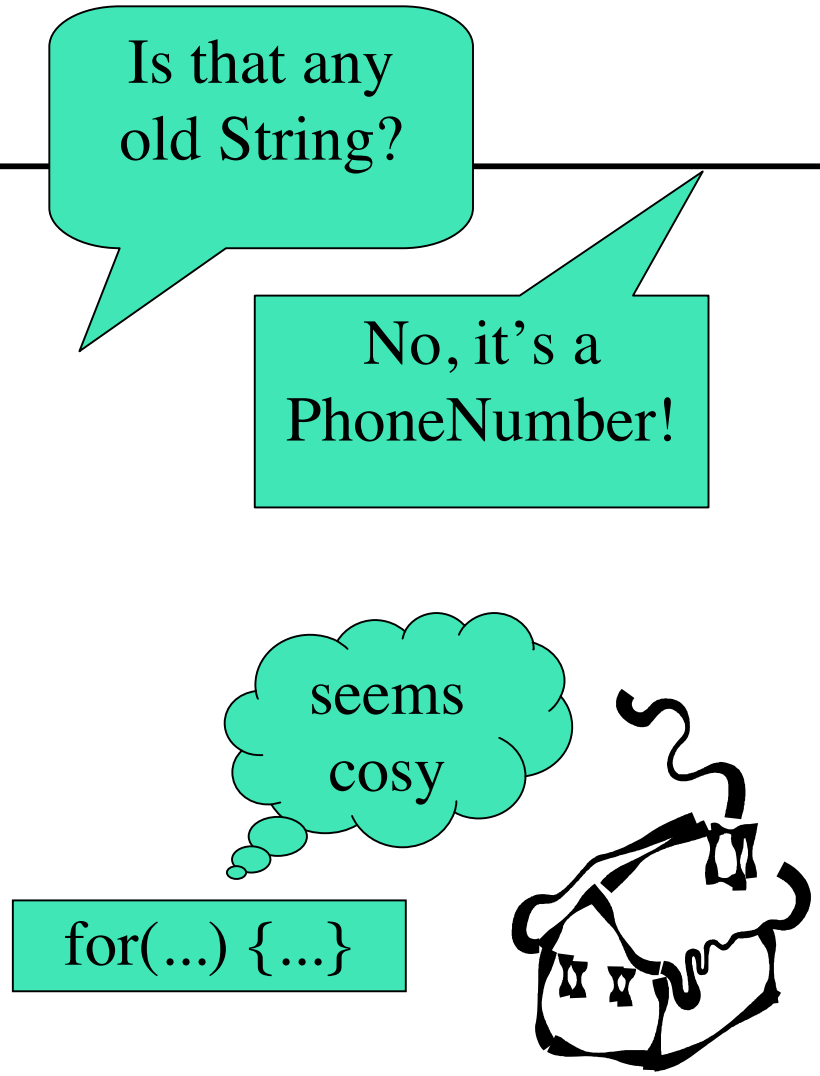
- Major drive of Domain Driven Design
- *Domain* Driven
  - Look for concepts in Domain
  - Let code follow suite
- Enrich language with new concept
  - Glossary
  - Code

Enter: Domain Logical Value Object, String Wrap Style

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

    public PhoneNumber(String number) {
        this.number = number;
    }

    public String getNumber() {
        return number;
    }
}
```

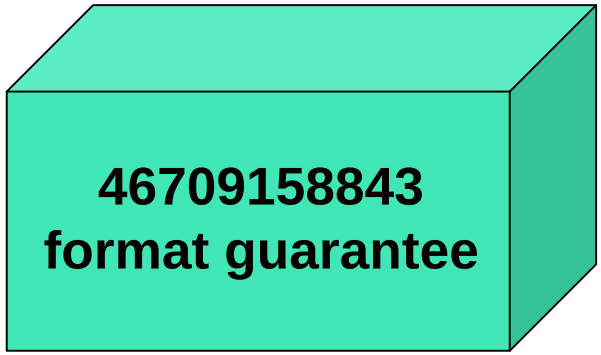
So What?

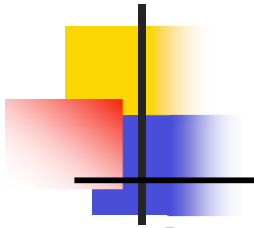
- New Type
  - understanding made explicit
  - code can talk about PhoneNumber
- New Class
  - where functionality can ”move in”



DLVO – with Format Control

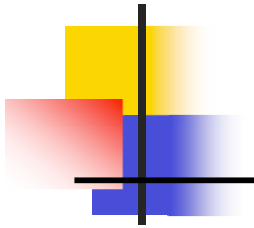
```
public class PhoneNumber {
    private final String number;

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

    public String getNumber() {
        return number;
    }

    static public boolean isValid(String number) {
        return ...;
    }
}
```





DLVO – Relieving BL Tier Code

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

    public PhoneNumber(String number) {
      ...
    }

    public String getNumber() { return number; }

    static public boolean isValid(String number) {
        return ...;
    }

    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) { ... }
}
```

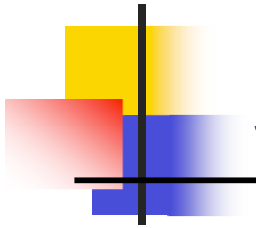


Evaluation Time

Did it get any better?

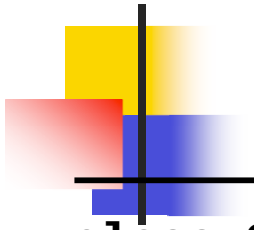
How about:

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



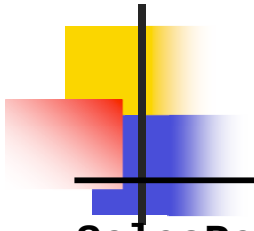
Service API Clarity

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



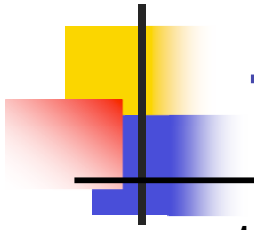
In-Data Validation and Error Handling

```
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();
```



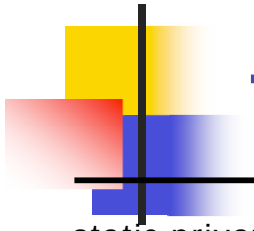
Focus of Business Logic Tier Code

```
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;
}
```



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*/ }
  }
```



Testability: Test code – CustomerService

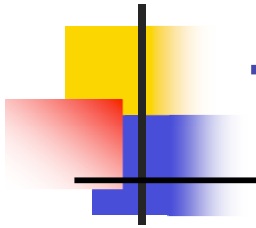
```
static private VALID_PHONE = new PhoneNumber("40068”)

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*/ }
   }
```

Number of tests

- 4 cases
- 3 uses

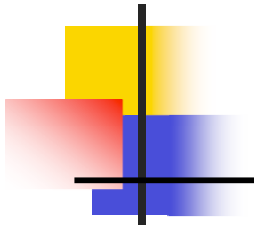
Total = m + n =
7 tests

	Phone Number	phone	fax	direct
null				
text				
empty				
plus				



Testability

- Lightweight testing
  - Need just PhoneNumber to test phonenumbers
  - Can be tested in lightweight environment
  - Blistering fast
- Does the format control always work?
  - better factored (separation of concerns)
  - fewer testcases m*n -> m+n
  - easy to do extensive



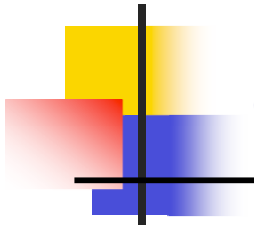
Bonus!

Note:

No changes to

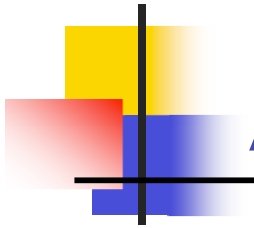
- Directory hierarchy
- Deployment routines
- Build scripts
- Classpath
- *etc*.

“Monday morning compliant!”



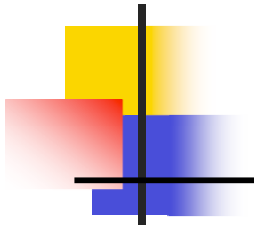
Candidates for DLVO

- Strings with format limitations
  - Name
  - Ordernumber
  - Zipcode
- Integers with limitations
  - Percentage (0-100%)
  - Quantity (>=0)
- Arguments/return values in service methods
  - Seldom “just data”
  - Data Transfer Objects for “multiple returns”
  - Exceptions can be domain logical



Anything worth remembering?

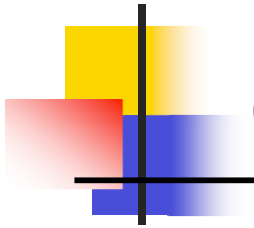
- Three points to remember tomorrow
-
-
-



Part Two – Power Use

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

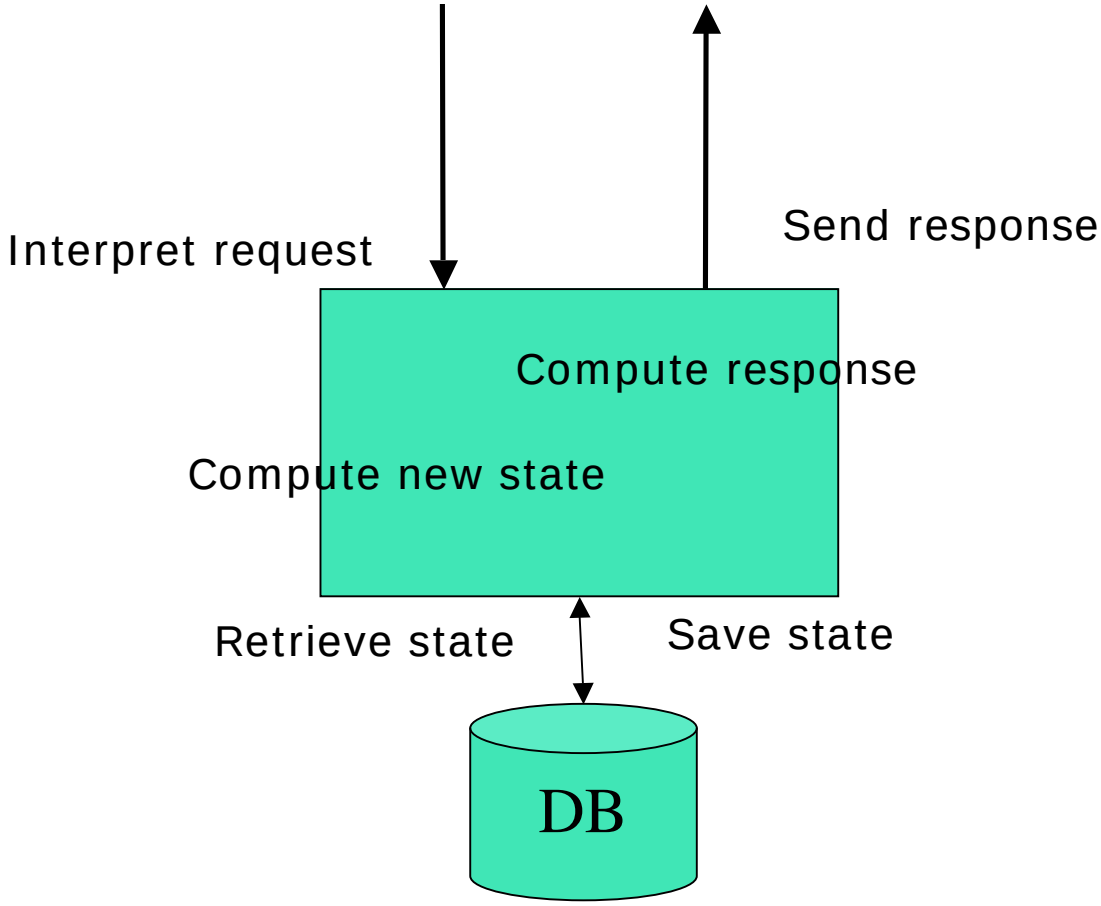
- Swiss dance instructor,
now living in San Francisco

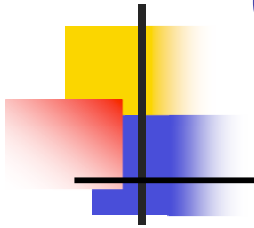


Get Value out of your Objects

Get value out of your objects by getting Values out of your (Service and Entity) Objects.

Things done on server

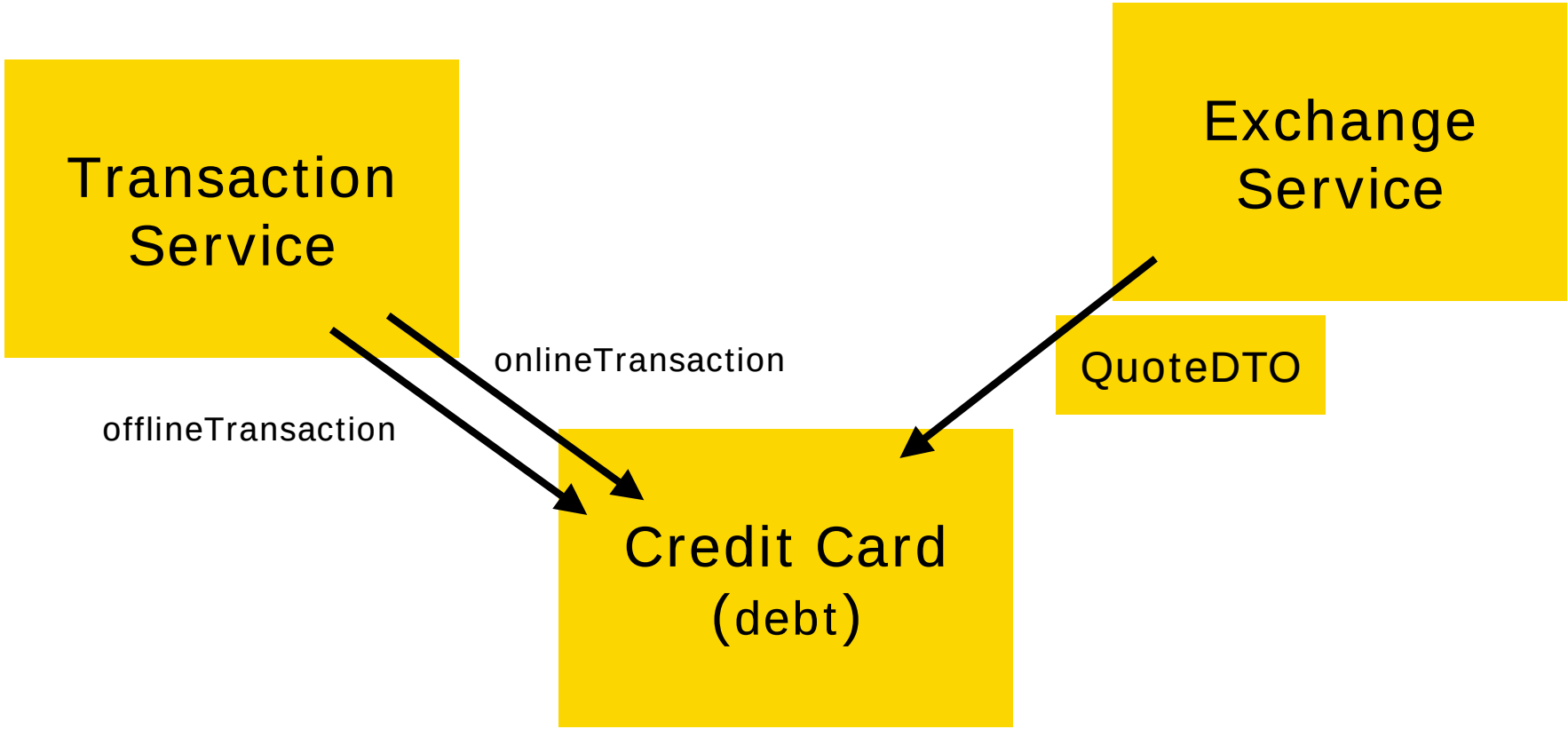


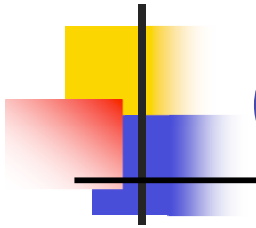


Credit Card and Multiple Currencies

- Card registered in one currency
- Card used with multiple currencies
- Online transaction
  - Reported immediately
  - Use current rate
- Offline transaction
  - Reported asynchronously
  - Use historical rate

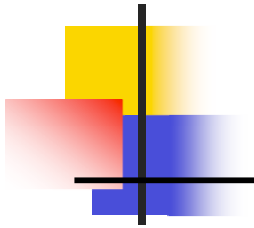
Architecture





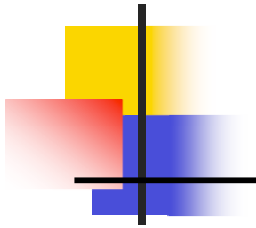
Credit Card Entity

```
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)
}
```



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;
}
```

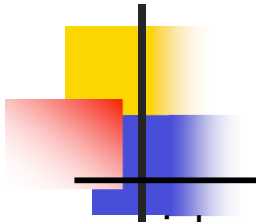


onlineTransaction(...)

```
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)));
        }
    }
}
```

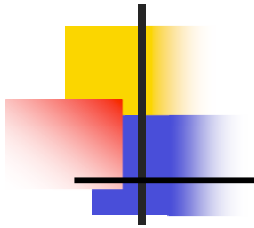
onlineTransaction(...)

```
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)));
        }
    }
}
```



offlineTransaction(...)

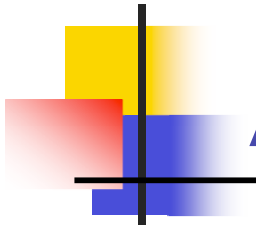
```
void offlineTransaction(StoreId store, Amount amount,
   Date purchaseday) {
   Currency storeCurrency = storeService.getCurrency(store);
    List<QuoteDTO> quotes = exchange.findRate(storeCurrency);
    QuoteDTO found = null;
    for (QuoteDTO quote : quotes) {
    if (quote.validfrom.before(purchaseday)
               && 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
}
```



Problem

Entity burdened with details

- Keeping track of currencies
- Performing exchange
- Quote validity



Afterburner

Compound value objects enter stage

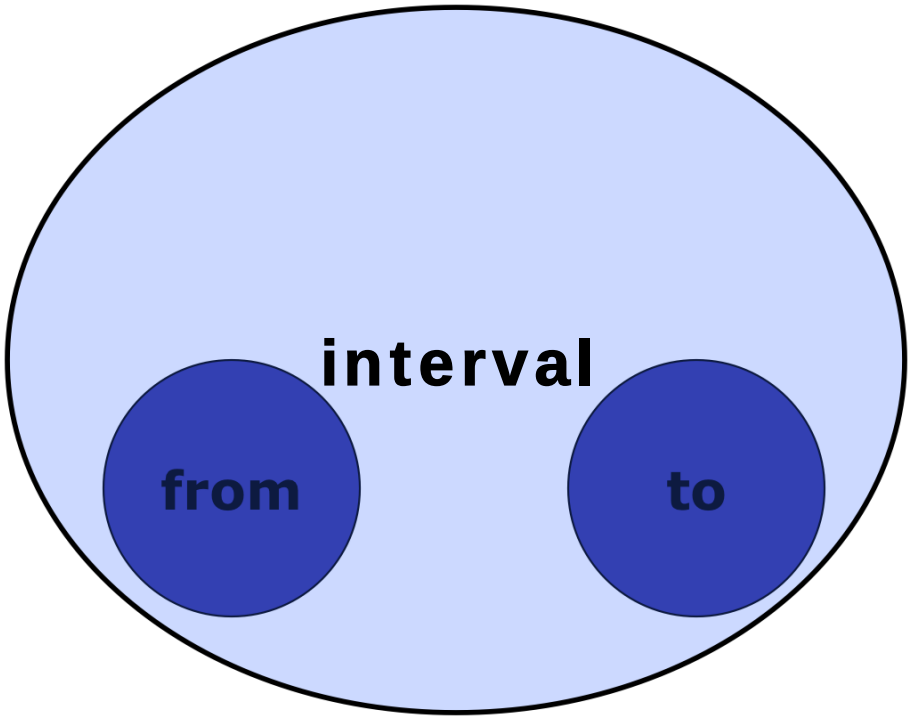
Buckle up

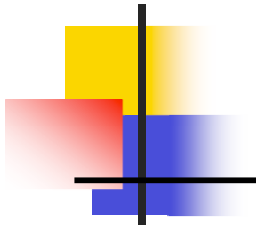
Prepare for take off

Refactoring: Encapsulate multi-object behaviour

- validfrom, validto, validity

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





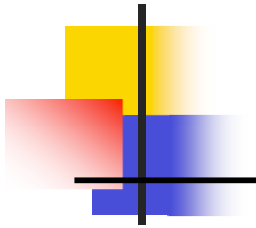
Refactoring: Introduce Data Pair Object

- Date + Date = TimeInterval

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

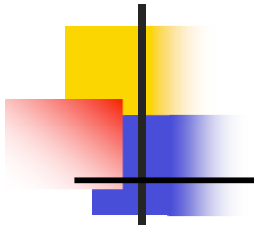
```
class TimeInterval {
   Date from;
   Date to;
   boolean contains(Date day) ...

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

- Why not QuoteDTO?



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



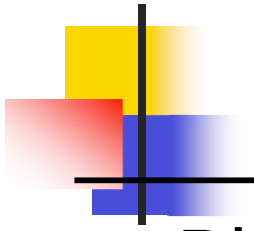
Make Context Explicit

- amount of what?

```
void offlineTransaction(StoreId store, BigDecimal amount,
   Date purchaseday) {
 Currency storeCurrency = storeService.getCurrency(store);

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

- Context in caller / surrounding

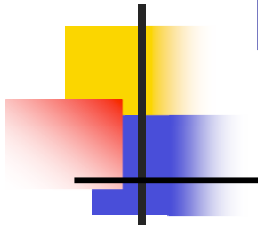
```
void debitCustomer( … ) {
    CreditCard card = cardReg.find(cardNumber);
    card.offlineTransaction(amount, date);
```



Make Context Explicit

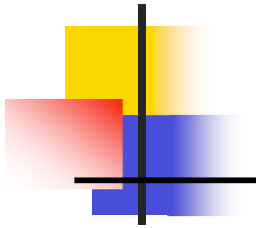
- BigDecimal + Currency = Money

```
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);
```



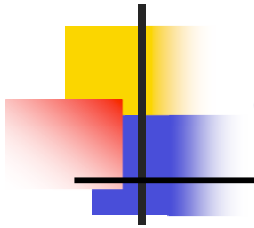
Data Transfer Object: Implicit context

```
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;
}
```



DTO – Explicit Context

```
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 from; // made explicit
    Currency to;
    BigDecimal rate;
    TimeInterval validinterval;
}
```



Compound Coherent Data

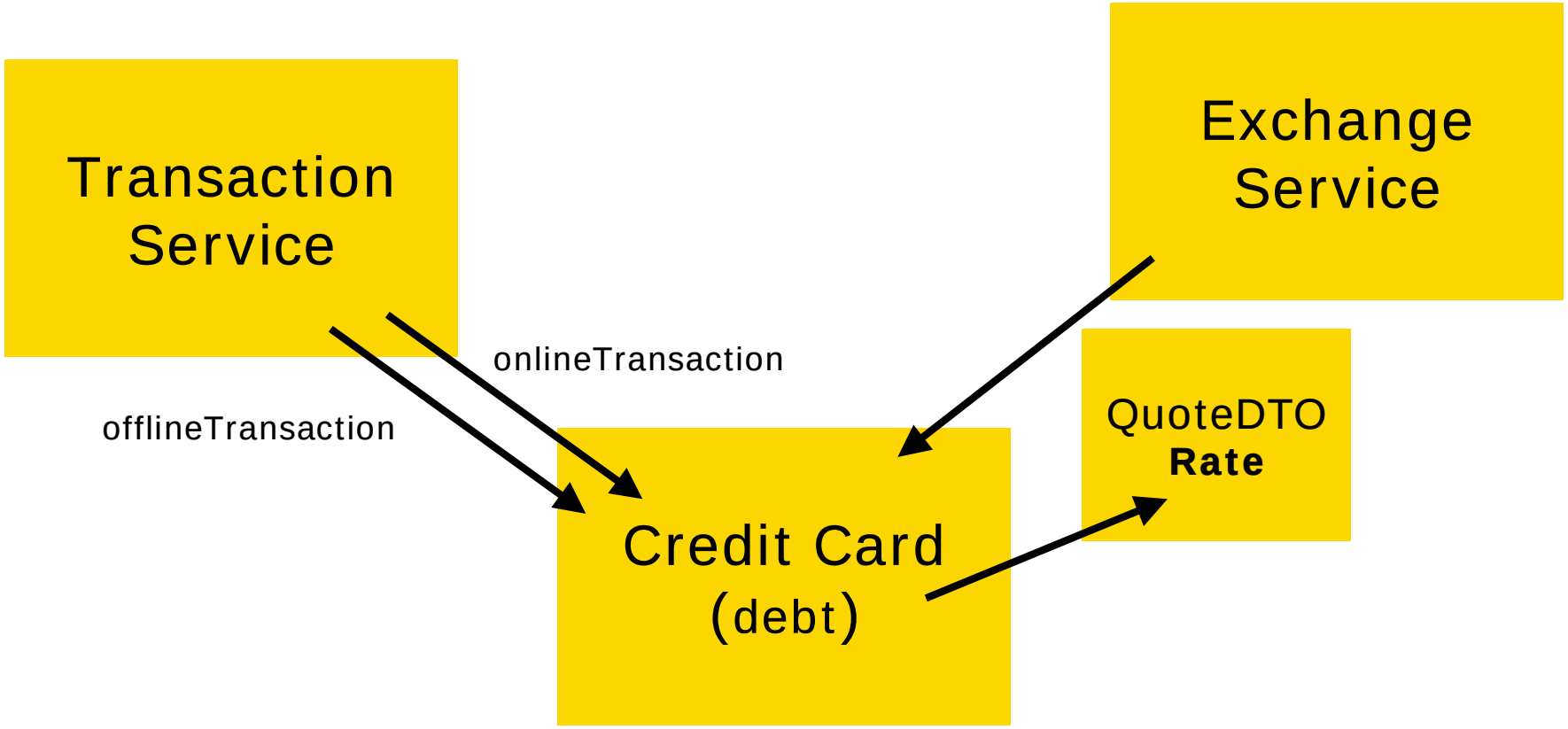
- from, to, rate – exchange logic

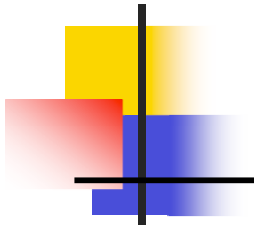
```
debt.add(
  amount.divide(cardquote.rate).multiply(storequote.rate))
```

- 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);

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

Architecture





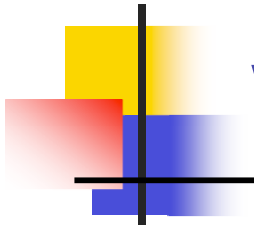
ExchangeService with smart Rates

ExchangeService returning DTO

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

ExchangeService returning Rates (VO)

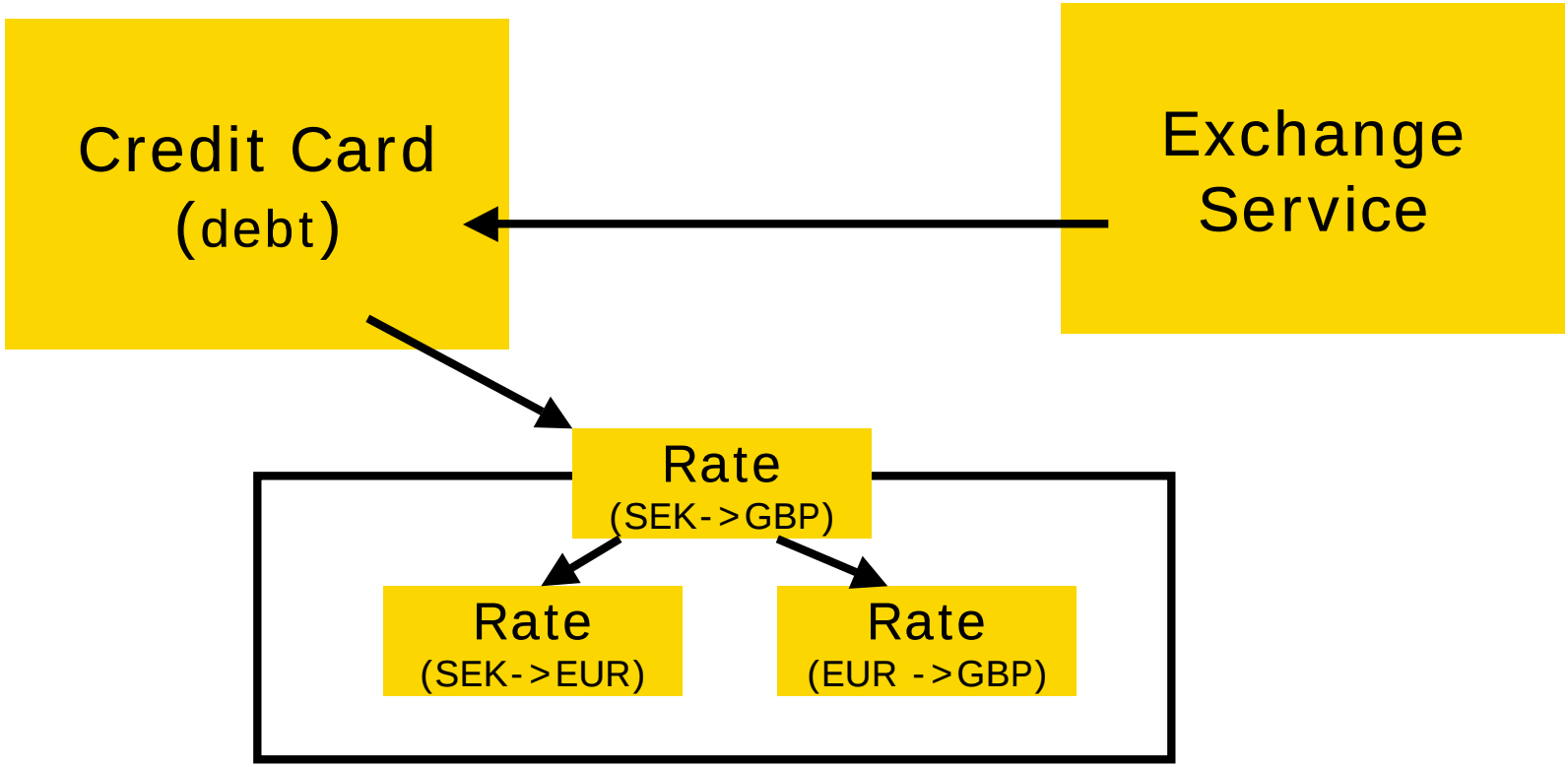
- Object with encapuslated behaviour
- Computations performed by returned object
- Can extend value objects with more behaviour

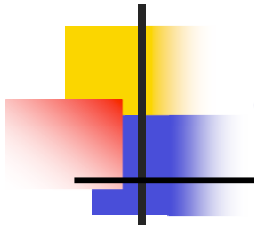


What about Fees?

- 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

... and Two-Step Exchange?

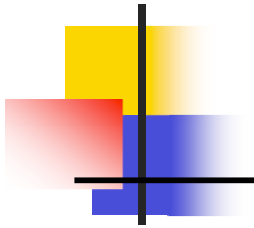




Composite Rate

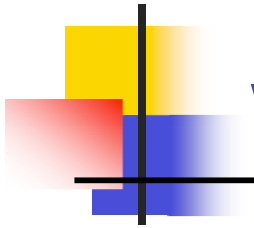
- SEK -> GBP
  - SEK -> REF_CURR
  - REF_CURR -> GBP

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



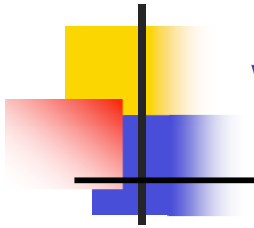
Finally DTO -> VO

```
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);
  }
}
```



What can we do with Quote VO?

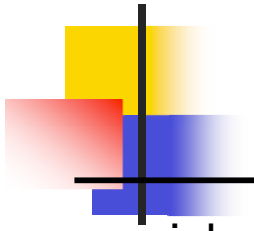
- How can we test?
- What can we test?
- MT-problem?
- How can we audit code?



What is Left?

onlineTransaction(...)

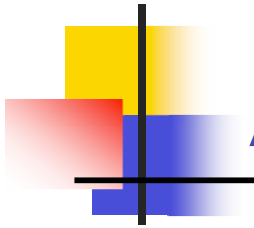
```
void onlineTransaction(StoreId store, BigDecimal amount) {
        Currency storeCurrency = storeService.getCurrency(store);
        if (storeCurrency.equals(cardcurrency)) {
            debt = debt.add(amount);
        } else if (cardcurrency.equals(ExchangeService.REF_CURR) &&
                (!storeCurrency.equals(ExchangeService.REF_CURR))){
            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)));
        }
    }
}
```



onlineTransaction(...)

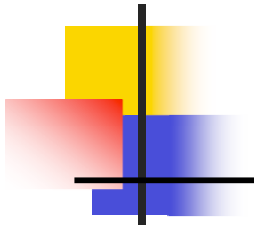
```
void onlineTransaction(Money m){
  Quote quote =
   exchange.findCurrentRate(m.getCurrency(), cardcurrency);
  debt = debt.add(quote.exchange(m, new Date());
}
```

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



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 (and Services) relieved of complexity
  - Uses advanced language
- Entity/Service code clearer

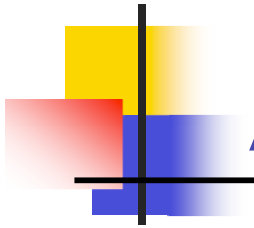


Recap – Overall Presentation Goal

Show how Value Objects can improve design and code clarity,

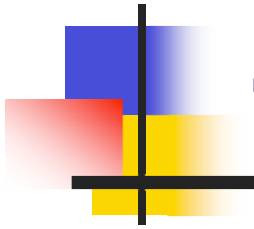
how to get started with them,

and some power usage



Anything worth remembering?

- Three points to remember tomorrow
-
-
-



</Power of Value>

Thanks for your attention

afterthoughts:

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

Colorado Software Summit