

# Apex Design Patterns

*Developers*





**Dennis Thong**

*Senior Technical Solution Architect*

**Richard Vanhook**

*Senior Technical Solution Architect*

# Safe Harbor

Safe harbor statement under the Private Securities Litigation Reform Act of 1995:

This presentation may contain forward-looking statements that involve risks, uncertainties, and assumptions. If any such uncertainties materialize or if any of the assumptions proves incorrect, the results of salesforce.com, inc. could differ materially from the results expressed or implied by the forward-looking statements we make. All statements other than statements of historical fact could be deemed forward-looking, including any projections of product or service availability, subscriber growth, earnings, revenues, or other financial items and any statements regarding strategies or plans of management for future operations, statements of belief, any statements concerning new, planned, or upgraded services or technology developments and customer contracts or use of our services.

The risks and uncertainties referred to above include – but are not limited to – risks associated with developing and delivering new functionality for our service, new products and services, our new business model, our past operating losses, possible fluctuations in our operating results and rate of growth, interruptions or delays in our Web hosting, breach of our security measures, the outcome of intellectual property and other litigation, risks associated with possible mergers and acquisitions, the immature market in which we operate, our relatively limited operating history, our ability to expand, retain, and motivate our employees and manage our growth, new releases of our service and successful customer deployment, our limited history reselling non-salesforce.com products, and utilization and selling to larger enterprise customers. Further information on potential factors that could affect the financial results of salesforce.com, inc. is included in our annual report on Form 10-Q for the most recent fiscal quarter ended July 31, 2012. This documents and others containing important disclosures are available on the SEC Filings section of the Investor Information section of our Web site.

Any unreleased services or features referenced in this or other presentations, press releases or public statements are not currently available and may not be delivered on time or at all. Customers who purchase our services should make the purchase decisions based upon features that are currently available. Salesforce.com, inc. assumes no obligation and does not intend to update these forward-looking statements.



# The Plan

Cover six design patterns

- Problem
- Pattern Abstract
- Code

Your participation is encouraged!



# Patterns

1. ?

2. ?

3. ?

4. ?

5. ?

6. ?



# Meet Billy, Your Admin



- 3+ yrs as Salesforce Admin
- Just recently started coding
- Sad because of this:

```
Trigger.AccountTrigger: line 3, column 1  
System.LimitException: Too many record  
type describes: 101
```

- Needs your help!



# The Offending Code

```
01 trigger AccountTrigger on Account (before insert, before update) {  
02     for(Account record : Trigger.new){  
03         AccountFooRecordType rt = new AccountFooRecordType();  
04         ....  
05     }  
06 }
```

```
07 public class AccountFooRecordType {  
08     public String id {get;private set;}  
09     public AccountFooRecordType() {  
10         id = Account.sObjectType.getDescribe()  
11             .getRecordTypeInfoByName().get('Foo').getRecordTypeId();  
12     }  
13 }
```

- When a single Account is inserted, what happens?
- When 200+ Accounts are inserted...?





# Solution: Singleton

Singleton
- instance : Singleton
- Singleton() + getInstance() : Singleton





# Let's Code It!

```
01 trigger AccountTrigger on Account (before insert, before update) {  
02     for(Account record : Trigger.new){  
03         AccountFooRecordType rt = new AccountFooRecordType();  
04         ....  
05     }  
06 }
```

```
07 public class AccountFooRecordType {  
08     public String id {get;private set;}  
09     public AccountFooRecordType() {  
10         id = Account.sObjectType.getDescribe()  
11             .getRecordTypeInfoByName().get('Foo').getRecordTypeId();  
12     }  
13 }
```

- Above is the offending code again
- Changes are highlighted in next slides



# Let's Code It!

```
01 trigger AccountTrigger on Account (before insert, before update) {  
02     for(Account record : Trigger.new){  
03         AccountFooRecordType rt = (new AccountFooRecordType()).getInstance();  
04         ....  
05     }  
06 }
```

```
07 public class AccountFooRecordType {  
08     public String id {get;private set;}  
09     public AccountFooRecordType() {  
10         id = Account.sObjectType.getDescribe()  
11             .getRecordTypeInfoByName().get('Foo').getRecordTypeId();  
12     }  
13     public AccountFooRecordType getInstance() {  
14         return new AccountFooRecordType();  
15     }  
16 }
```



# Let's Code It!

```
01 trigger AccountTrigger on Account (before insert, before update) {  
02     for(Account record : Trigger.new){  
03         AccountFooRecordType rt = AccountFooRecordType.getInstance();  
04         ....  
05     }  
06 }
```

```
07 public class AccountFooRecordType {  
08     public String id {get;private set;}  
09     public AccountFooRecordType() {  
10         id = Account.sObjectType.getDescribe()  
11             .getRecordTypeInfoByName().get('Foo').getRecordTypeId();  
12     }  
13     public static AccountFooRecordType getInstance() {  
14         return new AccountFooRecordType();  
15     }  
16 }
```



# Static

- Can modify member variables, methods, and blocks
- Executed when?
  - When class is introduced (loaded) to runtime environment
- How long does that last in Java?
  - Life of the JVM
- In Apex?
  - Life of the transaction



# Let's Code It!

```
01 trigger AccountTrigger on Account (before insert, before update) {  
02     for(Account record : Trigger.new){  
03         AccountFooRecordType rt = AccountFooRecordType.getInstance();  
04         ....  
05     }  
06 }
```

```
07 public class AccountFooRecordType {  
08     public String id {get;private set;}  
09     public AccountFooRecordType() {  
10         id = Account.sObjectType.getDescribe()  
11             .getRecordTypeInfoByName().get('Foo').getRecordTypeId();  
12     }  
13     public static AccountFooRecordType getInstance() {  
14         return new AccountFooRecordType();  
15     }  
16 }
```



# Let's Code It!

```
01 trigger AccountTrigger on Account (before insert, before update) {  
02     for(Account record : Trigger.new){  
03         AccountFooRecordType rt = AccountFooRecordType.getInstance();  
04         ....  
05     }  
06 }
```

```
07 public class AccountFooRecordType {  
08     public static AccountFooRecordType instance = null;  
09     public String id {get;private set;}  
10     public AccountFooRecordType() {  
11         id = Account.sObjectType.getDescribe()  
12             .getRecordTypeInfoByName().get('Foo').getRecordTypeId();  
13     }  
14     public static AccountFooRecordType getInstance() {  
15         if(instance == null) instance = new AccountFooRecordType();  
16         return instance;  
17     }  
18 }
```



# Let's Code It!

```
01 trigger AccountTrigger on Account (before insert, before update) {  
02     for(Account record : Trigger.new){  
03         AccountFooRecordType rt = AccountFooRecordType.getInstance();  
04         ....  
05     }  
06 }
```

```
07 public class AccountFooRecordType {  
08     private static AccountFooRecordType instance = null;  
09     public String id {get;private set;}  
10     public AccountFooRecordType() {  
11         id = Account.sObjectType.getDescribe()  
12             .getRecordTypeInfoByName().get('Foo').getRecordTypeId();  
13     }  
14     public static AccountFooRecordType getInstance() {  
15         if(instance == null) instance = new AccountFooRecordType();  
16         return instance;  
17     }  
18 }
```





# Let's Code It!

```
01 trigger AccountTrigger on Account (before insert, before update) {  
02     for(Account record : Trigger.new){  
03         AccountFooRecordType rt = AccountFooRecordType.getInstance();  
04         ....  
05     }  
06 }
```

```
07 public class AccountFooRecordType {  
08     private static AccountFooRecordType instance = null;  
09     public String id {get;private set;}  
10     private AccountFooRecordType() {  
11         id = Account.sObjectType.getDescribe()  
12             .getRecordTypeInfoByName().get('Foo').getRecordTypeId();  
13     }  
14     public static AccountFooRecordType getInstance() {  
15         if(instance == null) instance = new AccountFooRecordType();  
16         return instance;  
17     }  
18 }
```



# Patterns

1. Singleton

2. ?

3. ?

4. ?

5. ?

6. ?



**geocode('moscone center')**  
*//=> 37.7845935,-122.3994262*



# Your Suggestion to Billy

```
01 public class GoogleMapsGeocoder{
02     public static List<Double> getLatLong(String address){
03         //web service callout of some sort
04         return new List<Double>{0,0};
05     }
06 }

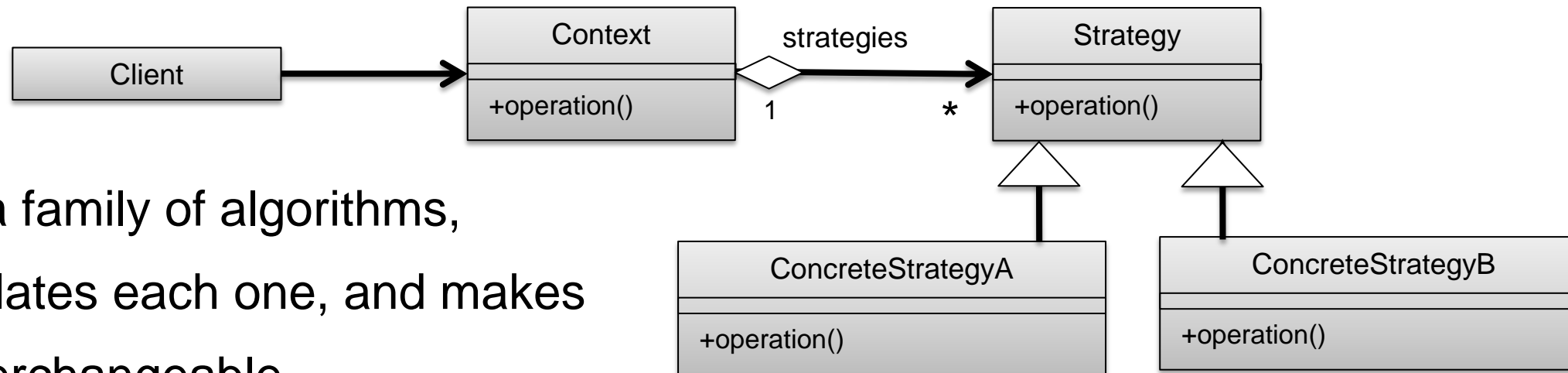
07 System.debug(GoogleMapsGeocoder.getLatLong('moscone center'));
08 //=> 13:56:36.029 (29225000) |USER_DEBUG|[29] |DEBUG| (0.0, 0.0)
```

## But then Billy mentions these requirements

- Need other options besides Google Maps
- Allow client code to choose provider



# Solution: Strategy



defines a family of algorithms,  
encapsulates each one, and makes  
them interchangeable

- Context => Geocoder
- operation() => getLatLong()
- Strategy => GeocodeService
- ConcreteStrategyA => GoogleMapsImpl
- ConcreteStrategyB => MapQuestImpl



# Let's Code It!

```
01 public interface GeocodeService{  
02     List<Double> getLatLong(String address);  
03 }
```

```
04 public class GoogleMapsImpl implements GeocodeService{  
05     public List<Double> getLatLong(String address){  
06         return new List<Double>{0,0};  
07     }  
08 }
```

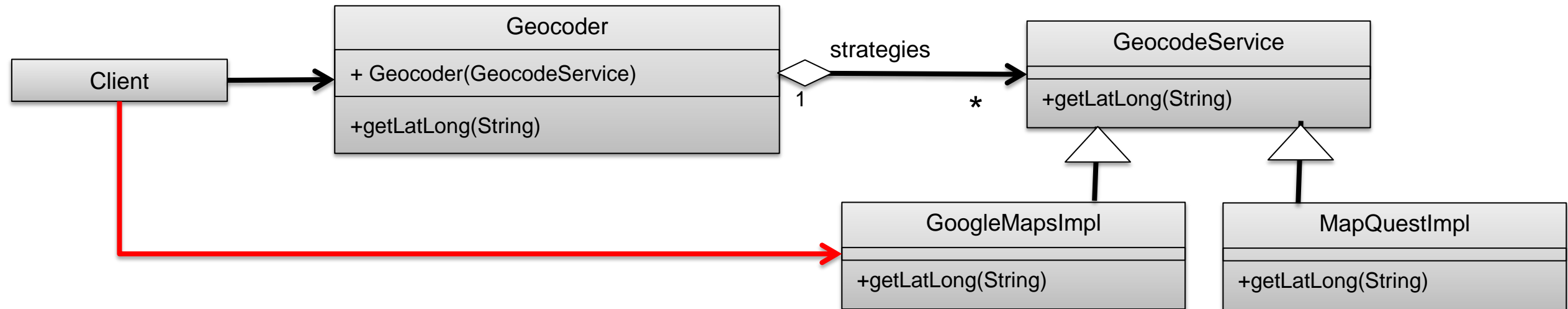
```
09 public class MapQuestImpl implements GeocodeService{  
10     public List<Double> getLatLong(String address){  
11         return new List<Double>{1,1};  
12     }  
13 }
```



```

01 public class Geocoder {
02     private GeocodeService strategy;
03     public Geocoder(GeocodeService s) {
04         strategy = s;
05     }
06     public List<Double> getLatLong(String address) {
07         return strategy.getLatLong(address);
08     }
09 }

```



```

10 Geocoder geocoder = new Geocoder(new GoogleMapsImpl());
11 System.debug(geocoder.getLatLong("moscone center"));
12 //=> 13:56:36.029 (29225000) |USER_DEBUG|[29] |DEBUG| (0.0, 0.0)

```

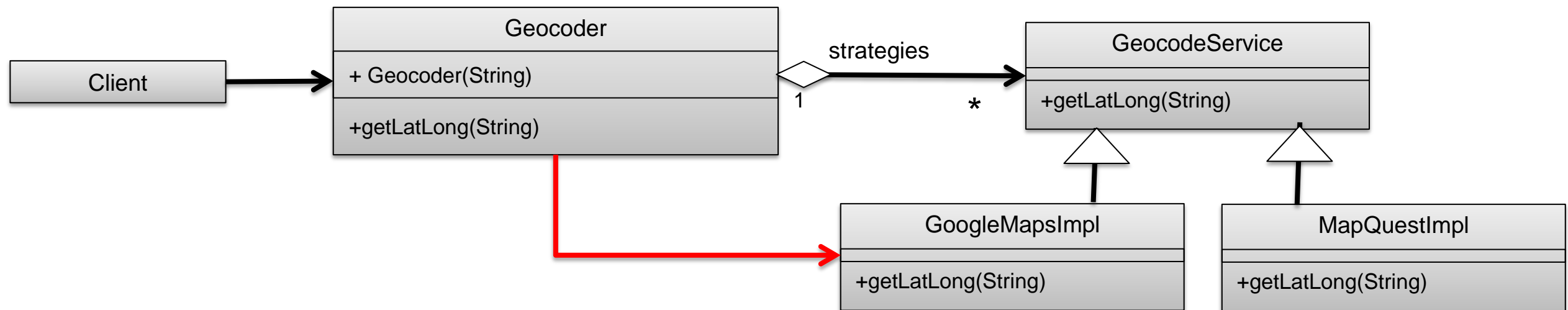




```

01 public class Geocoder {
02     public static final Map<String,Strategy> strategies = new
03         Map<String,Strategy>{'googlemaps' => new GoogleMapsImpl()
04         , 'mapquest'      => new MapQuestImpl()};
05     private GeocodeService strategy;
06     public Geocoder(String name){ strategy = strategies.get(name); }
07     public List<Double> getLatLong(String address){
08         return strategy.getLatLong(address);
09     }
10 }

```



```

11 Geocoder geocoder = new Geocoder('googlemaps');
12 System.debug(geocoder.getLatLong('moscone center'));
13 //=> 13:56:36.029 (29225000) |USER_DEBUG|[29]|DEBUG|(0.0, 0.0)

```



```

01 public class Geocoder {
02     public class NameException extends Exception{}
03     public static final Map<String,GeocodeService> strategies;
04     static{
05         GlobalVariable__c gv = GlobalVariable__c.getInstance('strategies');
06         List<String> strategyNames = new List<String>();
07         if(gv != null && gv.value__c != null) strategyNames = gv.value__c.split(',');
08         strategies = new Map<String,GeocodeService>();
09         for(String name : strategyNames){
10             try{strategies.put(name, (GeocodeService)Type.forName(name+'impl').newInstance());}
11             catch(Exception e){continue;} //skip bad name silently
12         }
13     }
14     private GeocodeService strategy;
15     public Geocoder(String name){
16         if(!strategies.containsKey(name)) throw new NameException(name);
17         strategy = strategies.get(name);
18     }
19     public List<Double> getLatLong(String address){
20         return strategy.getLatLong(address);
21     }
22 }

```



# Patterns

1. Singleton
2. Strategy
3. ?
4. ?
5. ?
6. ?



# Question?

How do we extend the functionality of an sObject in Apex without adding new fields?



**Search Duplicate Records**      Keyword:

	Person Name	Customer Name	Birthdate	Drivers License Number	Email	Preferred Phone
<input type="checkbox"/>	<u>Shana Clements</u>	<u>SHANE G CLEMENTS</u>		123456	<u>dthong_idmod@yahoo.com.au</u>	55555555
<input type="checkbox"/>	<u>Sharon Clements</u>	<u>SHANE G CLEMENTS</u>	1/01/1980	12345678		1234567890

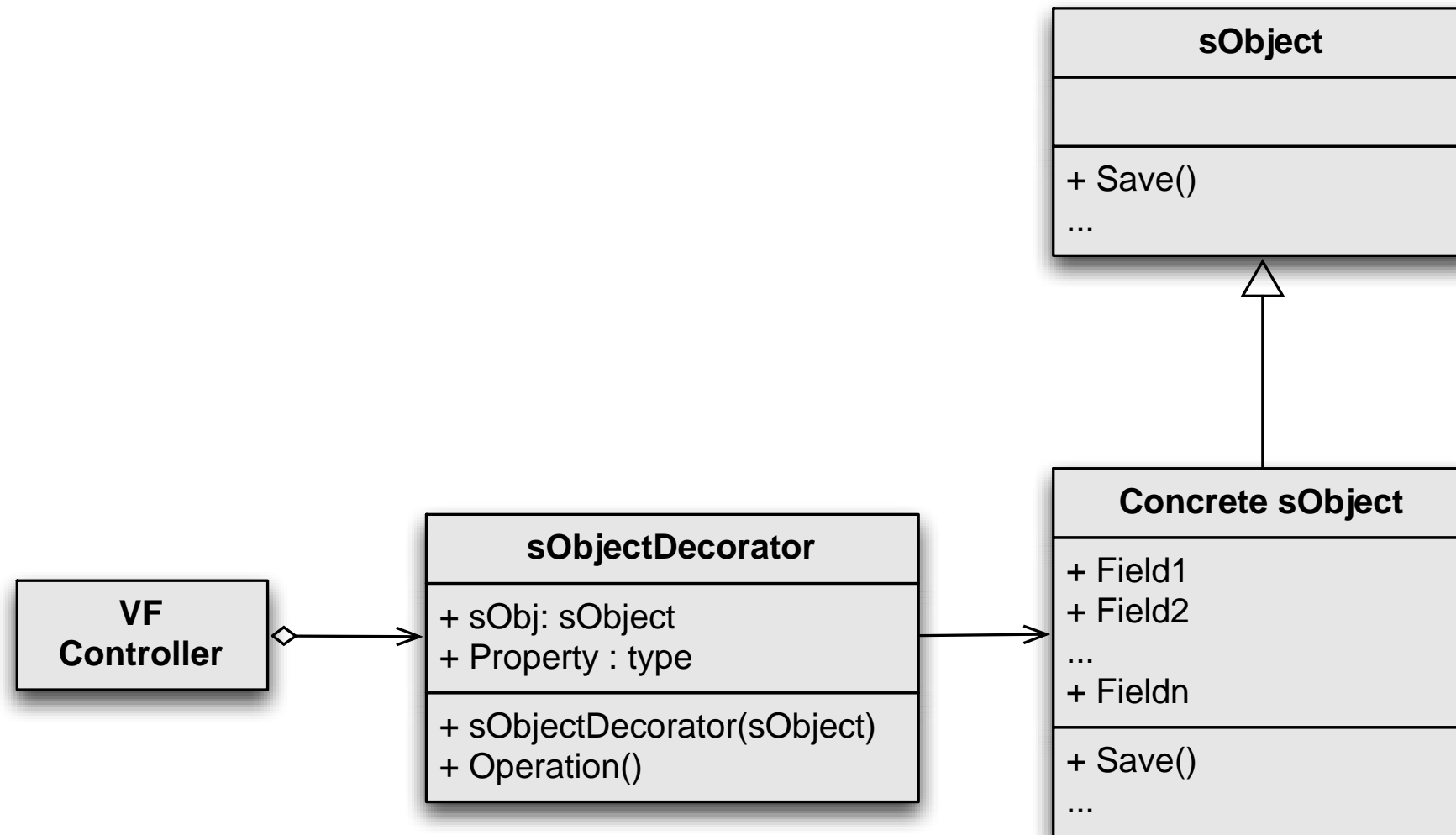
Transient selection checkboxes

City	Temperature (C)	Temperature (F)
Sydney	<input type="text" value="25.0"/>	77.00
Melbourne	<input type="text" value="19.0"/>	77.00
Brisbane	<input type="text" value="27.0"/>	80.60

Calculated Fields with Updates



# Solution – sObject Decorator



(aka Wrapper Classes)



#DF12



# Example - Scenario



## Weather sObject

- City\_\_c
- TempInFahrenheit\_\_c (in Fahrenheit)

## VisualForce Requirements

- Stored temperature in Fahrenheit is displayed in Celcius
- User enters Temperature in Celcius and is stored to the record in Fahrenheit

Save		
City	Temperature (C)	Temperature (F)
Sydney	<input type="text" value="25.0"/>	77.00
Melbourne	<input type="text" value="19.0"/>	77.00
Brisbane	<input type="text" value="27.0"/>	80.60

Bi-directional Display and Calculation



# The Code – Decorated sObject Class



```
01 public class DecoratedWeather {
02
03     public Weather__c weather { get; private set; }
04
05     public DecoratedWeather (Weather__c weather) {
06         this.weather = weather;
07     }
08
09     public Decimal tempInCelcius {
10         get {
11             if (weather != null && tempInCelcius == null )
12                 tempInCelcius = new Temperature().FtoC(weather.TempInFahrenheit__c);
13
14             return tempInCelcius;
15         }
16         set {
17             if (weather != null && value != null )
18                 weather.TempInFahrenheit__c= new Temperature().CtoF(value);
19
20             tempInCelcius = value;
21         }
22     }
23 }
```





# The Code – Custom Controller



```
01 public class Weather_Controller {
02
03     public List<DecoratedWeather> listOfWeather {
04
05         get {
06             if (listOfWeather == null) {
07                 listOfWeather = new List<DecoratedWeather>();
08
09                 for (Weather__c weather : [select name, temperature__c from Weather__c]) {
10                     listOfWeather.add(new DecoratedWeather(weather));
11                 }
12             }
13             return listOfWeather;
14         }
15
16         private set;
17     }
18 }
```



# The Code – VisualForce Page



```
01 <apex:page controller="weather_controller">
02
03     <!-- VF page to render the weather records with Ajax that only rerenders
04         the page on change of the temperature
05     -->
06     <apex:form id="theForm">
07         <apex:pageBlock id="pageBlock">
08             <apex:pageBlockTable value="{!listOfWeather}" var="weather">
09                 <apex:column value="{!weather.weather.name}"/>
10                 <apex:column headerValue="Temperature (C)">
11                     <apex:actionRegion >
12                         <apex:inputText value="{!weather.tempInCelcius}">
13                             <apex:actionSupport event="onchange"
14                                 reRender="pageBlock"/>
15                         </apex:inputText>
16                     </apex:actionRegion>
17                 </apex:column>
18                 <apex:column headerValue="Temperature (F)"
19                     value="{!weather.weather.Temperature__c}"
20                     id="tempInF"/>
21             </apex:pageBlockTable>
22         </apex:pageBlock>
23     </apex:form>
24 </apex:page>
```

No client side  
logic!!!



# Finally, on the VisualForce page

Save

City	Temperature (C)	Temperature (F)
Sydney	<input type="text" value="19.0"/>	66.20
Melbourne	<input type="text" value="19.0"/>	66.20
Brisbane	<input type="text" value="27.0"/>	80.60



# Patterns

1. Singleton
2. Strategy
3. sObject Decorater
4. ?
5. ?
6. ?



# What's wrong with this?



## VisualForce Controller #1

```
stub.timeout_x = 60000;
stub.endpoint_x = 'https://www.foo.com/createuser';

input.userid = Userinfo.getUserName();
input.timestamp = datetime.now();
input.Name = account.Name.toUpperCase();

CreateAccount_Service.Response response =
    stub.CreateAccount(input);
```

## VisualForce Controller #2

```
stub.timeout_x = 60000;
stub.endpoint_x = 'https://www.foo.com/createuser';

input.userid = Userinfo.getUserName();
input.timestamp = datetime.now();
input.Name = account.Name.toUpperCase();

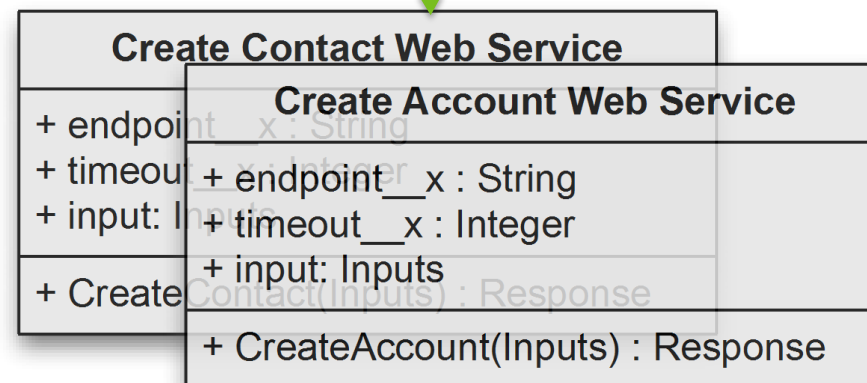
CreateAccount_Service.Response response =
    stub.CreateAccount(input);
```

## VisualForce Controller #3

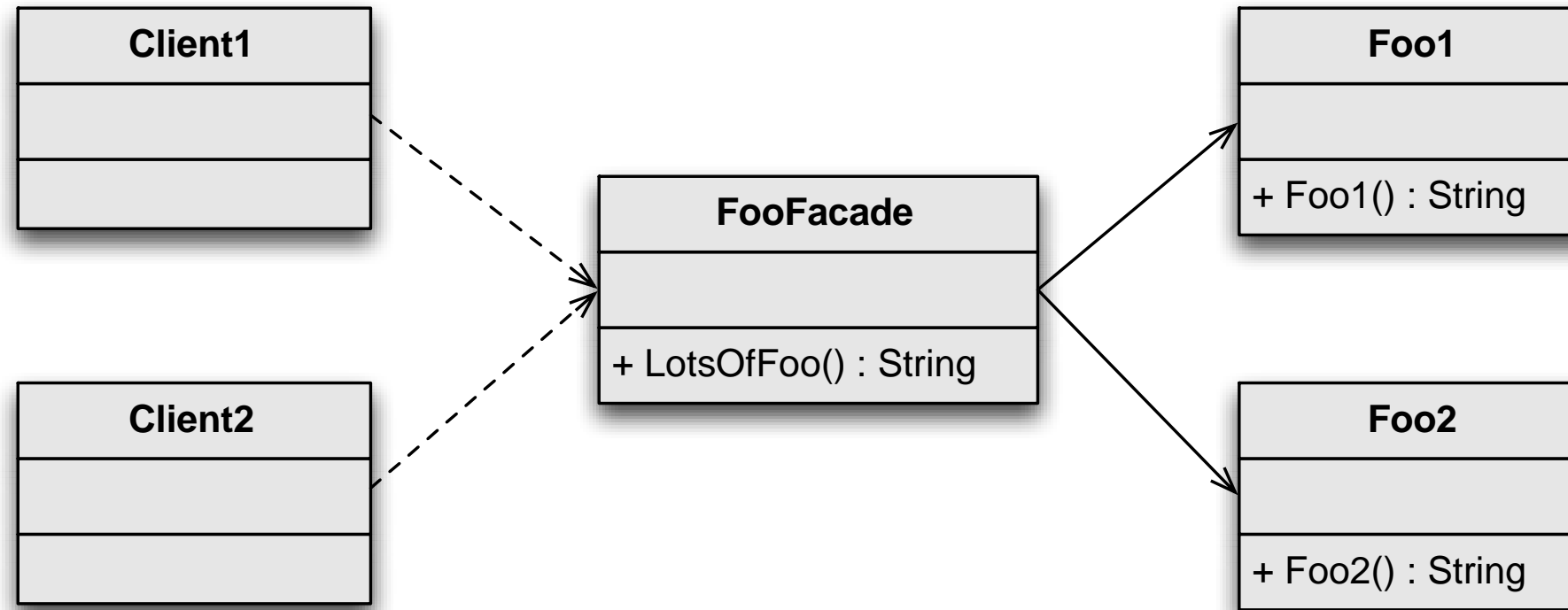
```
stub.timeout_x = 60000;
stub.endpoint_x = 'https://www.foo.com/createuser';

input.userid = Userinfo.getUserName();
input.timestamp = datetime.now();
input.Name = account.Name.toUpperCase();

CreateAccount_Service.Response response =
    stub.CreateAccount(input);
```



# Solution – Façade Pattern



```
01 public String LotsOfFoo () {
02     Foo1 f1 = new Foo1 ();
03     Foo2 f2 = new Foo2 ();
04
05     return f1.Foo1 () + f2.Foo2 ();
06 }
```



# Example - Scenario



## Composite Customer Create Transaction

- Create Account Web Service
- Create Contact Web Service

## Inputs

- UserId – current user
- Timestamp – current timestamp
- Account Name (Upper case transformation)
- Last Name and First Name

## Need to also set

- Timeout
- Hostname

Returns Account and Contact Number from remote system





# Code – Façade Class



```
01 public class CreateCustomerFacade {
02
03     public class CreateCustomerResponse {
04         public String accountNumber;
05         public String contactNumber;
06
07         public CreateCustomerResponse(String accountNumber,
08                                         String contactNumber) {
09             this.accountNumber = accountNumber;
10             this.contactNumber = contactNumber;
11         }
12     }
13     ...
```



# Code – Façade Class

```
01    ...
02    public String CreateCustomerExternal(String Name,
03        String LastName, String FirstName) {
04        CreateAccount_Service.CreateAccount stubCA =
05            new CreateAccount_Service.CreateAccount();
06        CreateAccount_Service.Inputs inputCA =
07            new CreateAccount_Service.Inputs();
08
09        stubCA.timeout_x = 60000;
10        stubCA.endpoint_x = 'https://www.foo.com/ca';
11
12        inputCA.userid = Userinfo.getUserName();
13        inputCA.timestamp = datetime.now();
14        inputCA.Name = name.toUpperCase();
15
16        String accountNumber = inputCA.CreateAccount(input);
17
18        /* REPEAT FOR CONTACT */
19
20        return new CreateCustomerResponse (
21            accountNumber, contactNumber);
22    }
```



# Code – The Client



```
01 public class FooController{
02
03     public Account account { get; set; }
04     public Contact contact { get; set; }
05     ...
06     public void CallCreateCustomerWS() {
07         CreateCustomerFacade ca =
08             new CreateCustomerFacade();
09
10         CreateCustomerFacade.CreateCustomerResponse resp =
11         ca.CreateCustomerExternal(account.name, contact.LastName,
12 contact.FirstName);
13
14         account.accountNumber = resp.accountNumber;
15         contact.contactNumber__c = resp.contactNumber;
    }
}
```



# Patterns

1. Singleton
2. Strategy
3. sObject Decorater
4. Façade
5. ?
6. ?



### Filter By Additional Fields (Optional):

Field	Operator	Value
1. Date	less than	9/18/2012
2. Owner Alias	equals	rvanhoo
3. Date	less than	8/18/2012
4. Owner Alias	equals	dthong
5. Days	equals	3

[Add Row](#) [Remove Row](#)

[Clear Filter Logic](#)

Filter Logic:

(1 AND 2) OR ((3 OR 4) AND 5)

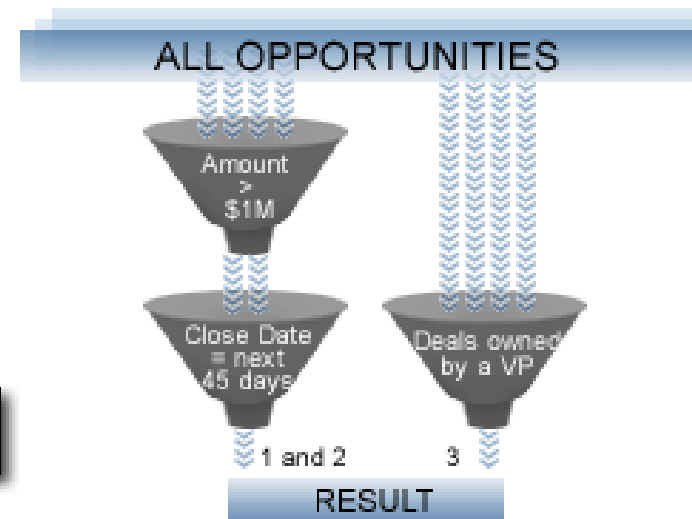
[Tips](#) ?

**Example:** If you wanted to filter to key deals for your company, where key deals are deals over \$1,000,000 that are closing in the next 45 days, or deals owned by a VP, you would set up your filters as follows

#### Advanced Filters:

Field	Operator	Value
1. Amount	greater than	1 000 000
2. Closed Date	equals	NEXT 45 DAYS
3. Owner Role	starts with	VP
4. --None--	equa	
5. --None--		

Advanced Filter Conditions:  
(1 AND 2) OR 3



# How do you represent expressions?

1 AND 2

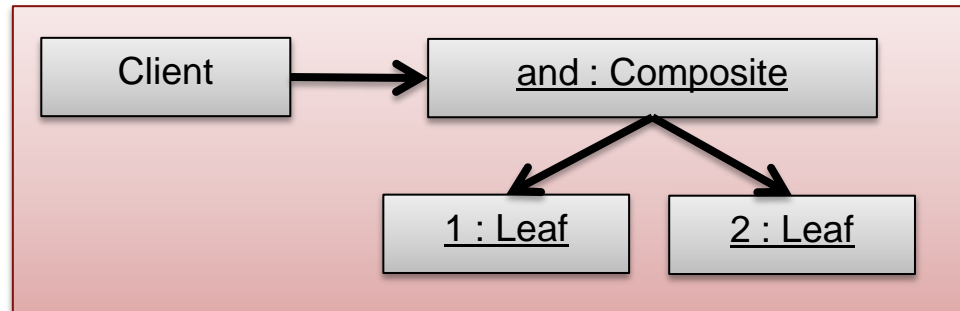
1 OR (2 AND 3)

(1 AND 2) OR ((3 OR 4) AND 5)

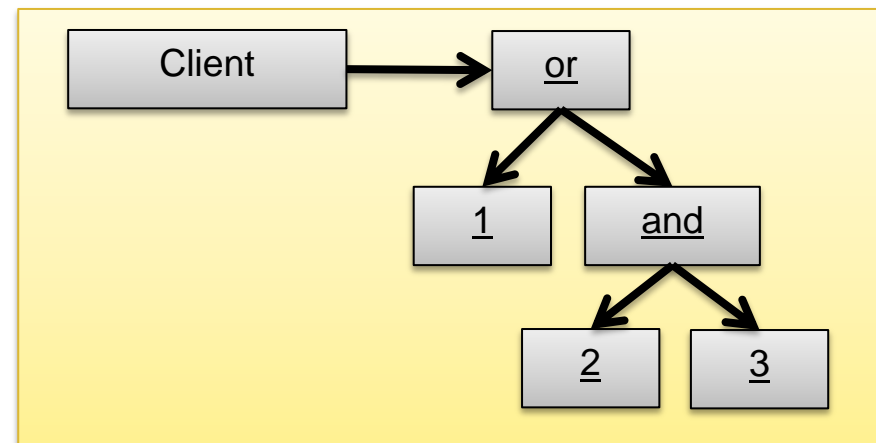


# Solution: Composite

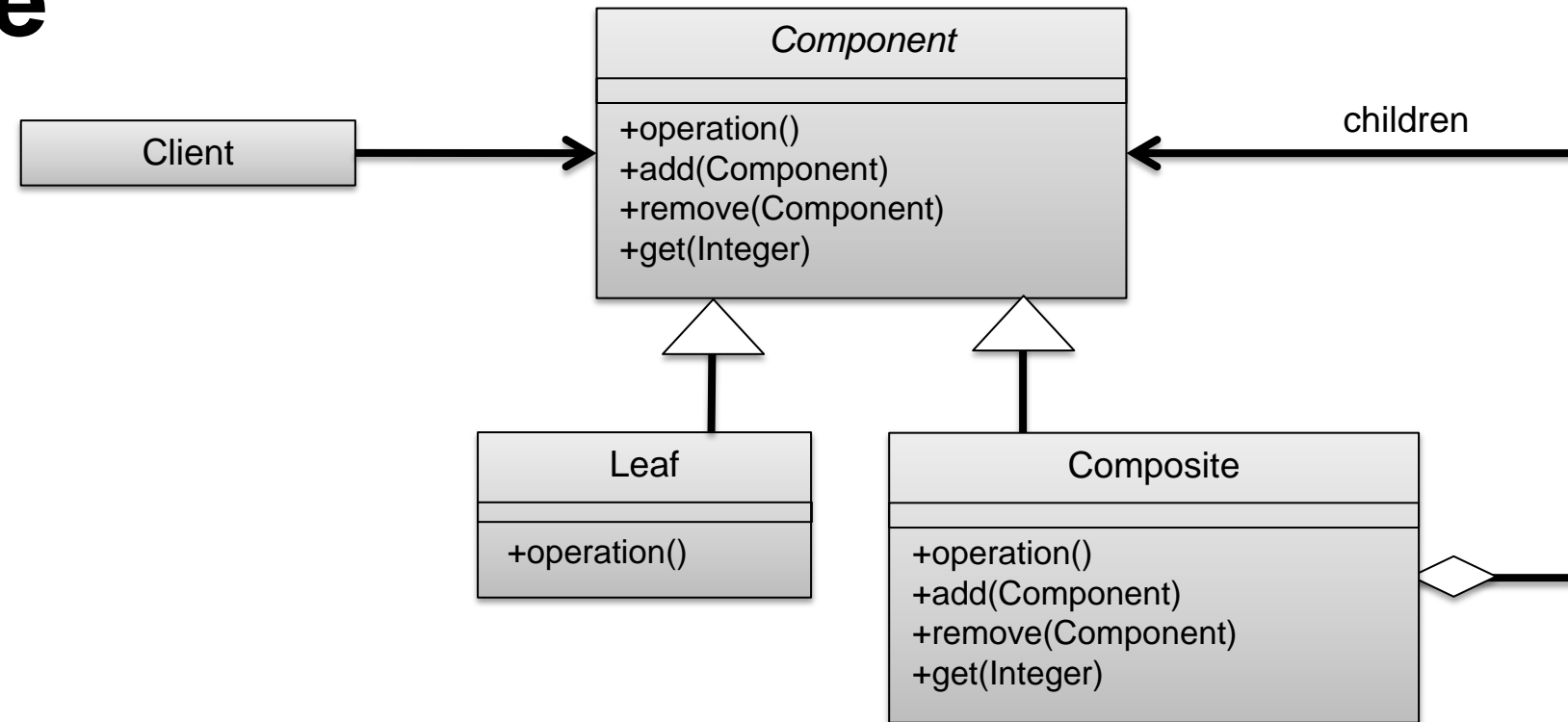
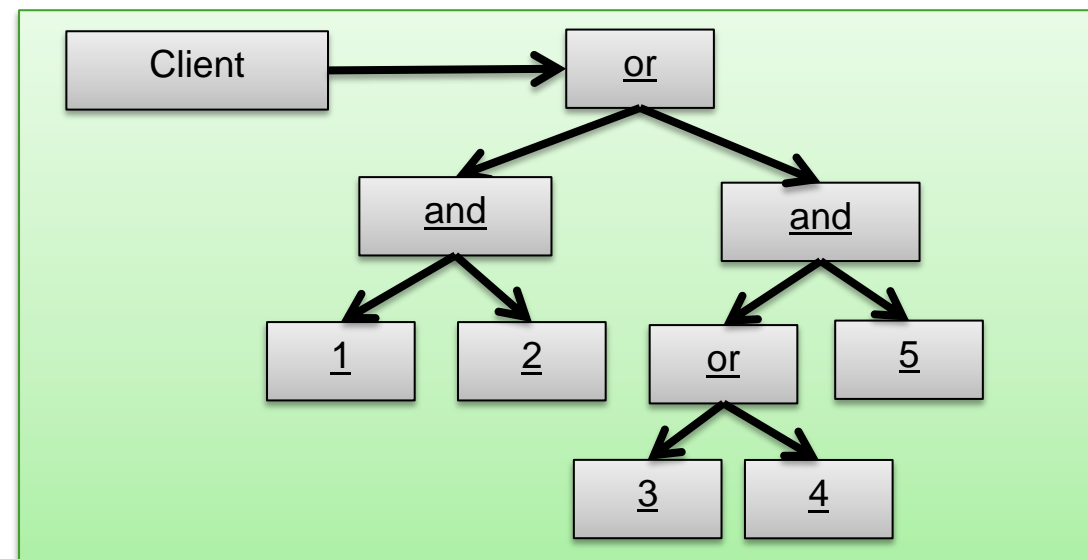
1 AND 2



1 OR (2 AND 3)



(1 AND 2) OR ((3 OR 4) AND 5)



# Let's Code It!

```
01 public interface Expression {  
02     Expression add(Expression expr);  
03     Expression set(String name, Boolean value);  
04     Boolean evaluate();  
05 }
```

```
06 public abstract class Composite implements Expression{  
07     public List<Expression> children {get; private set;}  
08     public Composite(){ this.children = new List<Expression>(); }  
09     public Expression add(Expression expr){  
10         children.add(expr); return this;  
11     }  
12     public Expression set(String name, Boolean value){  
13         for(Expression expr : children) expr.set(name,value);  
14         return this;  
15     }  
16     public abstract Boolean evaluate();  
17     public Boolean hasChildren{get{ return !children.isEmpty(); }}  
18 }
```



```
01 public class AndComposite extends Composite{
02     public override Boolean evaluate(){
03         for(Expression expr : children) if(!expr.evaluate()) return false;
04         return true;
05     }
06 }
```

```
07 public class OrComposite extends Composite{
08     public override Boolean evaluate(){
09         for(Expression expr : children) if(expr.evaluate()) return true;
10         return false;
11     }
12 }
```

```
13 public class Variable implements Expression{
14     public String name {get;private set;}
15     public Boolean value {get;private set;}
16     public Variable(String name){ this.name = name; }
17     public Expression add(Expression expr){ return this; }
18     public Expression set(String name, Boolean value){
19         if(this.name != null && this.name.equalsIgnoreCase(name))
20             this.value = value;
21         return this;
22     }
23     public Boolean evaluate(){ return value; }
24 }
```





# Building Expressions

```
01 //1 AND 2
02 Expression expr = new AndComposite();
03 expr.add(new Variable('1'));
04 expr.add(new Variable('2'));
```

```
05 //1 OR (2 AND 3)
06 Expression expr = new OrComposite();
07 expr.add(new Variable('1'));
08 Expression expr2 = new AndComposite();
09 expr.add(expr2);
10 expr2.add(new Variable('2'));
11 expr2.add(new Variable('3'));
```

```
12 //no need for expr2 var if using "method chaining"
13 //last line of add method: return this;
14 Expression expr = (new OrComposite())
15     .add(new Variable('1'))
16     .add((new AndComposite())
17         .add(new Variable('2'))
18         .add(new Variable('3'))
19     );
```



# Use Case

```
01 //1 OR (2 AND 3)
02 Expression expr = (new OrComposite())
03     .add(new Variable('1'))
04     .add((new AndComposite())
05         .add(new Variable('2'))
06         .add(new Variable('3'))
07     )
08     .set('1', false)
09     .set('2', true)
10     .set('3', false);
11
12 System.debug(expr.evaluate());
13 //FALSE OR (TRUE AND FALSE) => FALSE
14
15 expr.set('3', true);
16
17 System.debug(expr.evaluate());
18 //FALSE OR (TRUE AND TRUE) => TRUE
```



# Patterns

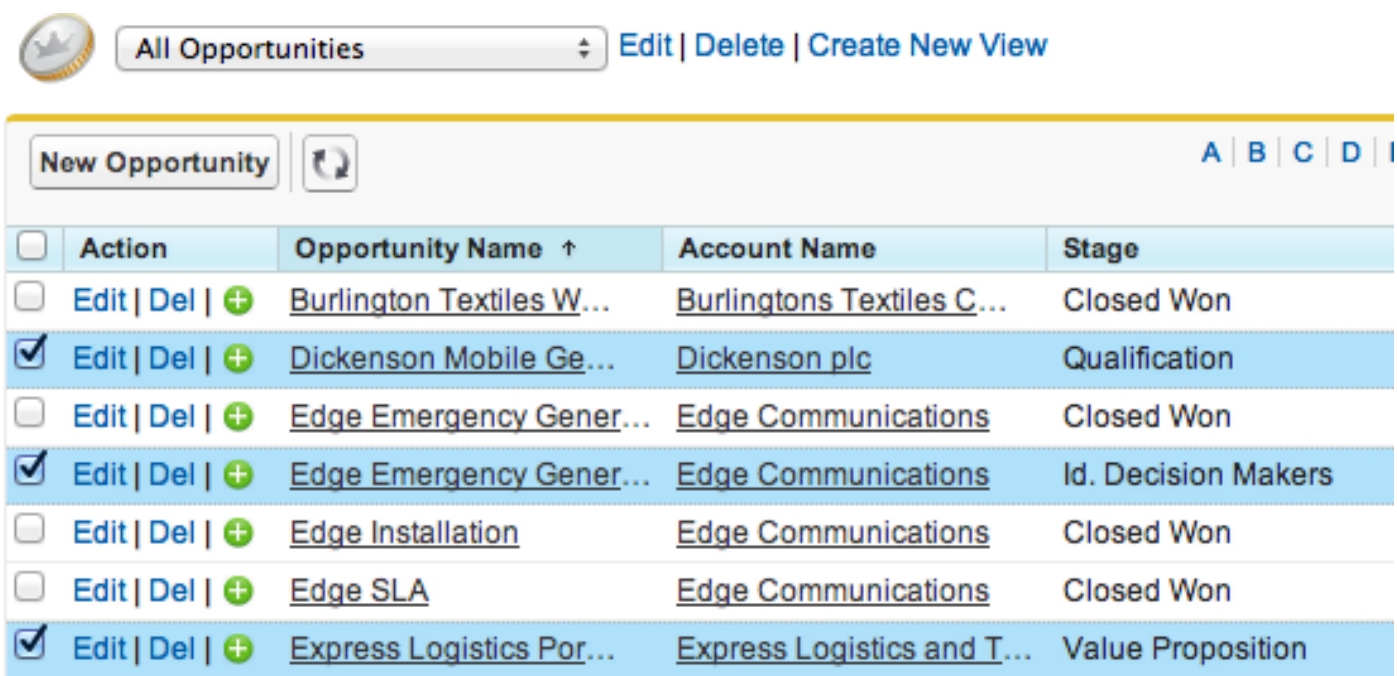
1. Singleton
2. Strategy
3. sObject Decorater
4. Façade
5. Composite
6. ?



# Billy Has a New Problem!

Billy wrote a trigger to create an order on close of an opportunity, however:

- It always creates a new order every time the closed opportunity is updated
- When loading via Data Loader, not all closed opportunities result in an order



UI elements above the table:   
 - Filter: All Opportunities   
 - Actions: Edit | Delete | Create New View   
 - Header: New Opportunity, Refresh icon, Sort icons (A, B, C, D, I)

<input type="checkbox"/>	Action	Opportunity Name ↑	Account Name	Stage
<input type="checkbox"/>	Edit   Del   +	<a href="#">Burlington Textiles W...</a>	<a href="#">Burlingtons Textiles C...</a>	Closed Won
<input checked="" type="checkbox"/>	Edit   Del   +	<a href="#">Dickenson Mobile Ge...</a>	<a href="#">Dickenson plc</a>	Qualification
<input type="checkbox"/>	Edit   Del   +	<a href="#">Edge Emergency Gener...</a>	<a href="#">Edge Communications</a>	Closed Won
<input checked="" type="checkbox"/>	Edit   Del   +	<a href="#">Edge Emergency Gener...</a>	<a href="#">Edge Communications</a>	Id. Decision Makers
<input type="checkbox"/>	Edit   Del   +	<a href="#">Edge Installation</a>	<a href="#">Edge Communications</a>	Closed Won
<input type="checkbox"/>	Edit   Del   +	<a href="#">Edge SLA</a>	<a href="#">Edge Communications</a>	Closed Won
<input checked="" type="checkbox"/>	Edit   Del   +	<a href="#">Express Logistics Por...</a>	<a href="#">Express Logistics and T...</a>	Value Proposition



# The Offending Code



```
01 trigger OpptyTrigger on Opportunity (after insert, after update) {  
02  
03     if (trigger.new[0].isClosed) {  
04         Order__c order = new Order__c();  
05         ...  
06         insert order  
07     }  
08 }
```

Occurs regardless  
of prior state

No Bulk  
Handling

Poor reusability



# Any Better?



```
01 trigger OpptyTrigger on Opportunity (after insert, after update) {  
02  
03     new OrderClass().CreateOrder(trigger.new);  
04  
05 }
```

```
01 public class OrderClass {  
02  
03     public void CreateOrder(List<Opportunity> opptyList) {  
04         for (Opportunity oppty : opptyList) {  
05             if (oppty.isClosed) {  
06                 Order__c order = new Order__c();  
07                 ...  
08                 insert order;  
09             }  
10         }  
11     }  
12 }  
13 }
```

Occurs regardless  
of prior state

Not bulkified



# How's this?



```
01 trigger OpptyTrigger on Opportunity (before insert, before update) {
02     if (trigger.isInsert) {
03         new OrderClass().CreateOrder(trigger.newMap, null);
04     else
0506         new OrderClass().CreateOrder(trigger.newMap, trigger.oldMap);
```

```
01 public class OrderClass {
02
03     public void CreateOrder(Map<Opportunity> opptyMapNew
04                             Map<Opportunity> opptyMapOld) {
05         List<Order__c> orderList = new List<Order__c>();
06         for (Opportunity oppty : opptyMapNew.values()) {
07             if (oppty.isClosed && (opptyMapOld == null ||
08                                     !opptyMapOld.get(oppty.id).isClosed)) {
09                 Order__c order = new Order__c();
10                 ...
11                 orderList.add(order);
12             }
13         }
14         insert orderList ;
15     }
```

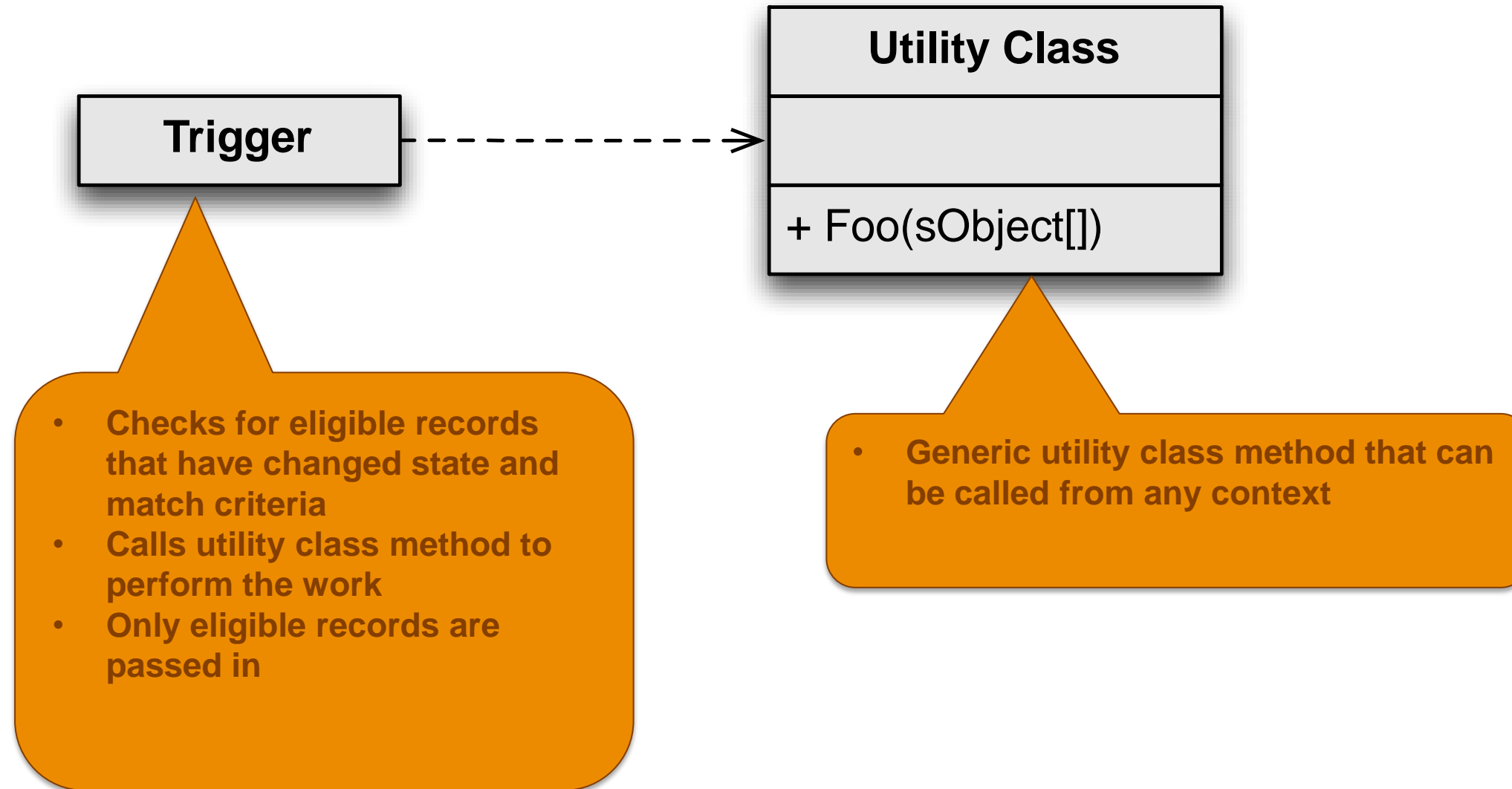
At least it's  
bulkified

This code is highly  
coupled and not reusable





# Solution – Bulk Transition





# At Last!!!



```
01 trigger OpptyTrigger on Opportunity (after insert, after update) {
02     if (trigger.isAfter && (trigger.isInsert || trigger.isUpdate)) {
03         List<Opportunity> closedOpptyList = new List<Opportunity>();
04         for (Opportunity oppty : trigger.new) {
05             if (oppty.isClosed && (trigger.isInsert ||
06                 (trigger.isUpdate &&
07                 !trigger.oldMap.get(oppty.id).isClosed)))
08                 closedOpptyList.add(oppty);
09         }
10         if (!closedOpptyList.isEmpty())
11             new OrderClass().CreateOrderFromOpptys(closedOpptyList)
```

Trigger handles state transition

```
01 public class OrderClass {
02     public void CreateOrdersFromOpptys(List<Opportunity> opptyList) {
03         List<Order__c> orderList = new List<Order__c>();
04         for (Opportunity oppty : opptyList) {
05             Order__c order = new Order__c();
06             ...
07             orderList.add(order);
08         }
09         insert orderList ;
```

This method is now a lot more reusable and is bulk-safe



# Patterns

1. Singleton
2. Strategy
3. sObject Decorater
4. Façade
5. Composite
6. Bulk State Transition



# Resources

Wrapper Classes - [http://wiki.developerforce.com/page/Wrapper\\_Class](http://wiki.developerforce.com/page/Wrapper_Class)

Apex Code Best Practices -

[http://wiki.developerforce.com/page/Apex\\_Code\\_Best\\_Practices](http://wiki.developerforce.com/page/Apex_Code_Best_Practices)

Apex Web Services and Callouts -

[http://wiki.developerforce.com/page/Apex\\_Web\\_Services\\_and\\_Callouts](http://wiki.developerforce.com/page/Apex_Web_Services_and_Callouts)

Sample Code - <https://github.com/richardvanhook/Force.com-Patterns>



# More at #DF12

## Hands-on Training: Introduction to Apex Patterns

Tuesday 3:30pm-6:00pm, Hilton Union Square, Continental Parlor 5

Thursday 12:00pm-2:30pm, Hilton Union Square, Continental Parlor 1/2/3



# Q&A



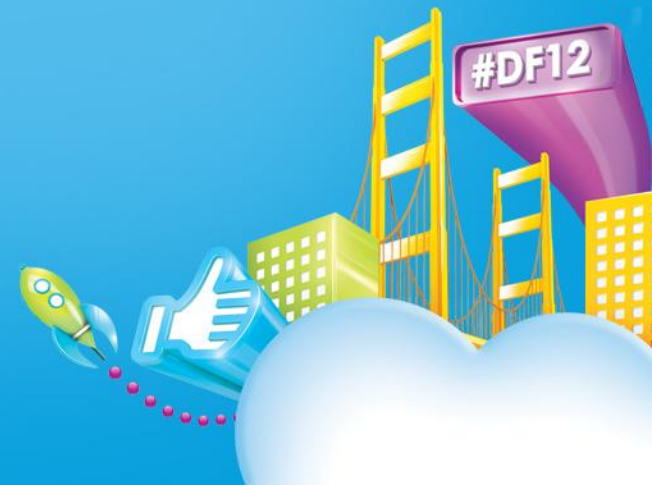
**Dennis Thong**

*Senior Technical Solution Architect*



**Richard Vanhook**

*Senior Technical Solution Architect*



# Thank You

salesforce®

