Writing Highly Genericized Code Using Dynamic Apex and the Tooling API

Jayvin Arora
Salesforce Practice Lead,
Salesforce MVP

@JayvinArora

Adam Olshansky
Salesforce Developer,
Salesforce MVP

AdamToArchitect.com
@adam17amo

Jayvin Arora

Salesforce Practice Lead,
Soliant Consulting

SOLIANT
CONSULTING®

SOLIANT CONSULTING

For more than a decade, our team has helped shape our clients' futures as well as served to simply make their business lives easier. Our role extends well beyond being experts in creating custom software – we consider ourselves your trusted business partner.

IMPLEMENTATION

As process consultants, we assist adoption by focusing on how your team works day-to-day.

APEX & VISUALFORCE

Our versatile, experienced team has a deep understanding of Salesforce's native languages.

APPEXCHANGE DEVELOPMENT

We develop AppExchange solutions and navigate the Salesforce security review process.

SYSTEMS INTEGRATION

Salesforce can seamlessly integrate with your existing business applications and IT infrastructure.

salesforce registered consulting partner

salesforce

Forward-Looking Statements

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 any litigation, risks associated with completed and any 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-K for the most recent fiscal year and in our quarterly report on Form 10-Q for the most recent fiscal quarter. These 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.

Goals

- An overview of Generic code for Apex
- Applying a design pattern in Apex
- Dynamic Apex
- Tooling API

Generic Programming in Apex, what's the scoop?

What problem are we solving for?

Rigid

Fragile

Poor Reusability

Lots of patches/quick fixes.

Duplicate code

Overly complicated data structures

- Maps of Maps of Maps of…..
- Managing the data in a 5-D Arrays

What are the solutions?

Process level

- Establish a definition of done.
- Good testing
- Code reviews
- Spending time to refactor code when adding new features
- Have clear coding decisions
- Have an architectural review when adding new features

What about the code structure?

Generic Programming

What is it?

When you take similar implementations of the same logic, and abstract them to a generic implementation that covers many use cases.

You can use the genericized implementation as the main plumbing of your process, and configure it for your individual cases.

Generic Programming

How do you do it?

Requires a good understanding of the problem space and the solution space

- Both are iterative processes

Having a structured approach to solving the problem

Identify the minimal requirements to the solution space

Generic Programming

How do you implement it?

Design Patterns

Ideally using Generics and Reflection. However, Apex, has a partial support of Generics and Reflection

Partial Support

1. Dynamic Apex/SOQL:
    a. sObjects -> Partial implementation of Generics
    b. Describes -> Partial implementation of Reflection

2. Non-sObjects - you end up with franken-code
    a. Tooling API (kinda sorta of…)
    b. Custom Types

Benefits of Generic Programming

Easier to reason about how the code functions, since the design is consistent:

- Makes it to add new features
- Easier to debug/ maintain
- Easier to onboard new team members

Less duplicate code

More Extensible & More Portable & More Reusable

Better Readability

This ultimately this leads to a better understanding of the problem and solution space.

Problem Statement

Imagine if you will….

1. Suppose, your company has leads coming in from all over the world,
2. Leads from different countries are routed to specific teams when they're inserted.
3. Then, each team has it's on own process to assign leads

Implementation

Declarative

- Each Lead Record has a Country field.
- A Custom Setting maps a Country to a Public Group
- Teams are represented by Public Groups which have Users as Group Members

Group
ASTRO

Edit | Delete

Label	ASTRO
Group Name	ASTRO
Grant Access Using Hierarchies	✓
Created By	Jayvin Arora, 10/5/2016 12:54 PM

View All Users

Name
Artie Ficial
Test User

Name ↑	Assigned Group
Afghanistan	ASTRO
Belgium	THUNDER
Canada	LIGHTNING
United States of America	LIGHTNING

```apex
/**
* checks if the lead is from a known country, and routes it to a
* default user, or public group associated with the country.
* Then the lead is assigned to a user based on that group's
* process.
* @param leadsToAssign are the list of leads that need to be assigned
* @return nothing
**/
public static void routeLead(List<Lead> leadsToAssign)
{
  //initialize variables
  List<Lead> defaultUserAssignments = new List<Lead>();
  List<Lead> astroAssignments = new List<Lead>();
  List<Lead> lightningAssignments = new List<Lead>();
  List<Lead> thunderAssignments = new List<Lead>();

  //get a map of known countries and the public group assigned to each country
  Map<String, Country_Team_Assignment__c> countryToAssignedTeams = Country_Team_Assignment__c.getAll();

  //assign leads to the proper team
  for (Lead leadRec : leadsToAssign)
  {
    //Is the lead from a known country?  No - assign to default user that's lazy loaded
    if(!countryToAssignedTeams.containsKey(leadRec.Country))
      leadRec.OwnerId = defaultUser.Id;
    else
    {
    //Route leads from a known country to the assigned group
      String countryGroup = countryToAssignedTeams.get(leadRec.Country).Assigned_Group__c;
      if (countryGroup == Constants.ASTRO)
        astroAssignments.add(leadRec);
      else if(countryGroup == Constants.LIGHTNING)
        lightningAssignments.add(leadRec);
      else if(countryGroup == Constants.THUNDER)
        thunderAssignments.add(leadRec);
    }
  }

  //assign lead records to users based on the team. Each time has a unique routing mechanism
  assignLeadstoAstroMembers(astroAssignments);
  assignLeadsToLightningMembers(lightningAssignments);
  assignLeadsToThunderMembers(thunderAssignments);
}
```

Code, continued

```
63
64  /**
65  * Loop through the leads and assign them to a Random Member
66  * If the Group has no members, we assign them to a default user.
67  * @param leadRecs the leadRecords assigned to the astro team
68  * @return nothing
69  **/
70  public static void assignLeadstoAstroMembers(List<Lead> leadRecs)
71  {
72    //get the astro team members. the map is keyed by team name, and is lazy loaded
73    List<Id> astroMembers = new List<Id>(groupsAssignedToCountries.get(Constants.ASTRO));
74    for(Lead leadRec : leadRecs)
75    {
76      //assign to default user
77      if (astroMembers.isEmpty())
78        leadRec.OwnerId = defaultUser.Id;
79      else
80      { //assign randomly
81        Integer randomNum = Math.roundToLong(Math.random()*(astroMembers.size() - 1)).intValue();
82        leadRec.OwnerId = astroMembers[randomNum];
83      }
84    }
85  }
```

A new requirement comes in...

The Lead routing process is so effective, that it needs to be extended to more groups and the same team-specific routing logic applied to cases

What do we do?

Design Patterns

Why Design Patterns?

Provides a structured template for the thinking about the problem and solution.

Allows for a higher level of abstraction which makes it more reusable

Makes architectures more accessible to others

Patterns are very well documented

Common Apex Design Patterns

Structural	Creational	Behavioral
Decorator	Singleton	**Strategy**
Facade	Factory	Bulk State Transition
Composite		
Adapter		

Design Pattern- Strategy

What is it?

A pattern that allows you to encapsulate a collection of algorithms and choose one at runtime.

When to use them?

A client defines many behaviors, with multiple conditional statements in its operations.

When you have a list of similar business processes that you want to implement

To avoid exposing complex, logic to your client process.

Strategy Implementation

So I can use design patterns like legos to solve problems?

Patterns are NOT legos

Approach for using Patterns

- Identify the problem you're trying to solve
- How do I modify/ simplify the patterns to solve your use case.
- Don't try to force a design pattern on a problem

Tradeoffs to using/adopting design patterns

Higher learning curve - build in time for teaching
More Classes/objects

Dynamic Apex

Tools of the trade

- Field Sets
- Custom Settings/Metadata
- Dynamic SOQL
- Limits class
- Schema classes
- sObjects

AstroGenericRouter.cls

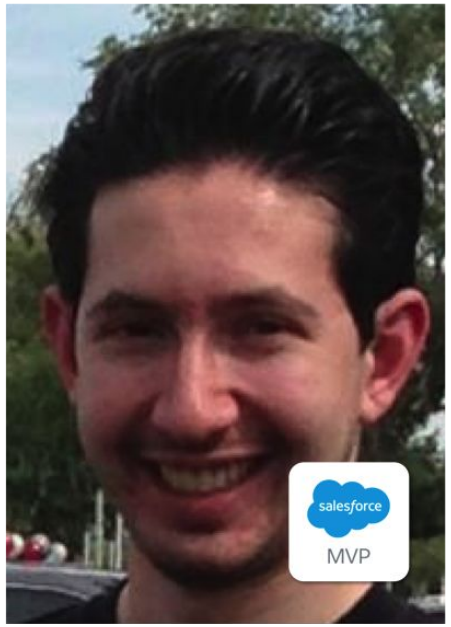
```
1  public class AstroGenericRouter extends Router {
2
3    // custom routing algorithm for astro public group
4    public override void assign(List<SObject> records){
5
6
7      //get the astro team members. the map is keyed by team name, and is lazy loaded
8      List<Id> astroMembers = new List<Id>(groupsAssignedToCountries.get(Constants.ASTRO));
9      Integer membersSize = astroMembers.size();
10     for(SObject rec : records)
11     {
12       //assign to default user
13       if (astroMembers.isEmpty())
14         rec.put('OwnerId', defaultUser.Id);
15       else
16       { //assign randomly
17         Integer index = Integer.valueOf(Math.random()* membersSize);
18         rec.put('OwnerId', astroMembers[index]);
19       }
20     }
21   }
22 }
```

```
1.  public class AstroRouter extends Router {
2.
3.          // custom routing algorithm for astro public grou
    p
4.          public override void assign(List<SObject> record
    s){
5.                  List<Lead> leads = (List<Lead>) records;
6.
7.                  //get the astro team members. the map is
     keyed by team name, and is lazy loaded
8.                  List<Id> astroMembers = new List<Id>(grou
    psAssignedToCountries.get(Constants.ASTRO));
9.                  Integer membersSize = astroMembers.size
    ();
10.                 for(Lead lead : leads)
11.                 {
12.                         //assign to default user
13.                         if (astroMembers.isEmpty())
14.                                 lead.OwnerId = defaultUse
    r.Id;
15.                         else
16.                         { //assign randomly
17.                                 Integer index = Integer.v
    alueOf(Math.random()* membersSize);
18.                                 lead.OwnerId = astroMembe
    rs[index];
19.                         }
20.                 }
21.         }
22. }
```

```
1.  public class AstroGenericRouter extends Router {
2.
3.          // custom routing algorithm for astro public grou
    p
4.          public override void assign(List<SObject> record
    s){
5.
6.
7.                  //get the astro team members. the map is
     keyed by team name, and is lazy loaded
8.                  List<Id> astroMembers = new List<Id>(groupsAssign
    edToCountries.get(Constants.ASTRO));
9.                  Integer membersSize = astroMembers.size
    ();
10.                 for(SObject rec : records)
11.                 {
12.                         //assign to default user
13.                         if (astroMembers.isEmpty())
14.                                 rec.put('OwnerId', defaul
    tUser.Id);
15.                         else
16.                         { //assign randomly
17.                                 Integer index = Integer.v
    alueOf(Math.random()* membersSize);
18.                                 rec.put('OwnerId', astroM
    embers[index]);
19.                         }
20.                 }
21.         }
22. }
```

Adam Olshansky

Salesforce Developer,
eTouch Systems at YouTube



eTouch

Tooling API

An API for building...TOOLS

Tooling API Highlights

Great for building...tools!

- Gather finely grained metadata about Apex classes, triggers, and tests in an org
  - Classes, triggers, construction of symbol tables
  - Query for code coverage, logs, SOQL results
- Query metadata about setup objects
  - Business Processes, Flows, Custom items, Workflow components
- Can be utilized for source control/continuous integration
  - Metadata containers, deployment details, code file metadata
- Can be queried through REST or SOAP
- Similar to Metadata API but with a ton of additional functionality

Our Scenario

UTILIZE DYNAMIC APEX TO WRITE APEX USING APEX!

APEX-CEPTION!!!

Additional Tooling API Use Cases

- Show all objects that can be analyzed
  - /services/data/v37.0/tooling/sobjects/
- Show list of all classes
  - /services/data/v37.0/tooling/apexManifest
- Show all metadata about a class
  - /services/data/v37.0/tooling/sobjects/ApexClass/01po0000004SL5b

https://andyinthecloud.com/category/tooling-api/

Tools To Be Built

- UML Diagram Tool
- Symbol Table Visualizer
- Automated Testing Tools?
- Tools to identify unused code?
- Customize managed packages based on org's config
- Use your imagination!

Where to Learn More

- Follow @andyinthecloud and his blog!
- Andy Fawcett's Blog https://andyinthecloud.com/category/tooling-api/
- Andy and James Loghry's Apex Tooling API Wrapper: https://github.com/afawcett/apex-toolingapi/
- Pat Patterson's Java Tooling API Wrapper: https://github.com/metadaddy/force-tooling-demo https://developer.salesforce.com/blogs/developer-relations/2013/01/new-in-spring-13-the-tooling-api.html
- Tom Patros VF page Utilizing Symbol Tables: https://github.com/tompatros/df13-tooling-api-demo

Summary

Next Steps from Here

Summary and Next Steps

- Be thinking about the right type of code for the app you're building
- Utilize design patterns
- Write dynamic code
- Gather granular details to build any app
- Get out there and build!

Thank You

References

https://developer.salesforce.com/page/Apex_Design_Patterns
Gamma, E, et al, 1994 Design Patterns, Elements of Reusable Object-Oriented Software

Information about sObjects

```
01 // sObject types to describe
02 String[] types = new String[]{'Account','Merchandise__c'};
03
04 // Make the describe call
05 Schema.DescribeSobjectResult[] results = Schema.describeSObjects(types);
06
07 System.debug('Got describe information for ' + results.size() + ' sObjects.');
08
09 // For each returned result, get some info
10 for(Schema.DescribeSobjectResult res : results) {
11     System.debug('sObject Label: ' + res.getLabel());
12     System.debug('Number of fields: ' + res.fields.getMap().size());
13     System.debug(res.isCustom() ? 'This is a custom object.' : 'This is a standard object.');
14     // Get child relationships
15     Schema.ChildRelationship[] rels = res.getChildRelationships();
16     if (rels.size() > 0) {
17         System.debug(res.getName() + ' has ' + rels.size() + ' child relationships.');
18     }
19 }
```

Dynamic SOQL

Query at RUN time instead of COMPILE time

Query on Record Type Only If It Exists

Database.query()

- Don’t always know what parameters are available in a user’s org
- In your org this works fine

```
List<Lead> rtypes = [SELECT ID, RecordTypeID FROM Lead LIMIT 1];
```

- What if your customer doesn’t have record types defined for Lead

```
Boolean leadHasRecordType =
Schema.SObjectType.Lead.Fields.getMap().containsKey('recordtypeid');

String fieldString = 'ID ';
if(leadHasRecordType){
        fieldString += ', RecordTypeId ';
}

List<Lead> rtypes = Database.Query('Select ' + fieldstring + ' from Lead LIMIT 1');
```

Thanks to Dan Appleman, Advanced Apex Programming

Dynamic Visualforce

Dynamic Controller

```
01 public with sharing class bookExtension {
02     private ApexPages.StandardController controller;
03
04     private Set<String> bookFields = new Set<String>();
05
06     public bookExtension (ApexPages.StandardController controller) {
07         this.controller = controller;
08         Map<String, Schema.SobjectField> fields =
09         Schema.SobjectType.Book__c.fields.getMap();
10
11         for (String s : fields.keySet()) {
12         // Only include accessible fields
13             if (fields.get(s).getDescribe().isAccessible() &&
14                 fields.get(s).getDescribe().isCustom()) {
15                     bookFields.add(s);
16             }
17         }
18     }
19
20     public  List<String> availableFields {
21         get {
22             controller.reset();
23             controller.addFields(new List<String>(bookFields));
24                 return new List<String>(bookFields);
25         }
26     }
27 }
```

Dynamic Page using $ObjectType

```
01 <apex:page standardController="Book__c" extensions="bookExtension" >
02
03     <br/>
04     <apex:pageBlock title="{!Book__c.Name}">
05         <apex:repeat value="{!availableFields}" var="field">
06
07             <h2><apex:outputText
08                 value="{!$ObjectType['Book__c'].Fields[field].Label}"/></h2>
09             <br/>
10             <apex:outputText value="{!Book__c[field]}" /><br/><br/>
11
12         </apex:repeat>
13     </apex:pageBlock>
14
15 </apex:page>
```

Limits Class

Dynamically adjust code based on limits

Query the Limits Class

Dynamically make adjustments based on limits

- Limits.getDMLRows();
- Limits.getDMLStatements();
- Limits.getHeapSize();
- Limits.getQueries();
- Limits.getQueryRows();
- bit.ly/apexlimitsclass (Apex Limits Class)

Tradeoffs

Negatives

- More code to write
- More code to test
- Less readability

Positives

- More detailed information
- Less chance of an error down the road
- Endless possibilities

OLD SLIDES

Types of Design Patterns in OOP

Three families of Design Patterns

- Creational
- Structural
- Behavioral

Design Patterns: Elements of Reusable Object-Oriented Software -Erich Gamma,Richard Helm,Ralph Johnson, John Vlissides (Gang of 4)

Who are we talking to today?

- ISV developers
  - Don’t know all conditions of customer orgs
- Individual app developers
  - May need to accept input from clients/customers
- Any developer accepting user input
  - Tailor your queries and results based on user conditions

2 Different Ways of Writing Code

Static

```
List<Account> accounts = [SELECT ID, Name, Segment__c, Value__c FROM Account
WHERE Name IN :acctSet AND Value__c < 5000];
return accounts;
```

Dynamic/Generic

```
String query = 'SELECT ID, Name, Segment__c, Value FROM Account WHERE ';

String names = '(' + enteredNames + ')';
query += 'Name IN ' + names;

String value = '';
if(enteredValue != null){
        value = String.valueOf(enteredValue);
        query += ' AND Value__c < ' + value;
}

System.debug(query);
List<Account> accounts = Database.query(query);
return accounts;
```

What are the benefits of generic code?

Design Patterns

- Tried and tested solutions to work with
- Aid communication and well documented
- Have a proven track record as they are already widely used and thus reduce the technical risk to the project

Dynamic Apex/SOQL & Tooling API

- Highly flexible and can be used in practically any type of application or domain
- Create SOQL queries at runtime with Apex code
- Write generic and dynamic VF pages programmatically
- Removes metadata dependencies

Different Ways of Writing Code: Static vs. Dynamic

Static	Dynamic/Generic
Generated at compile time	Generated at run time
Hard coded	Dependent on user input
Doesn’t usually have dependencies	May have org-specific dependencies
Works well if you know what is in the org	Works well if you DON’T know what’s in the org
Most commonly used by developers at a company developing for their own org	Most commonly used by developers at ISVs whose code needs to be able to run in a number of differently customized orgs
Field id dependencies can be managed	Manages field Id dependencies

Tying it Together

- You know how to design apps efficiently
- You know how to write generic code
- You know how to find information about org metadata
- What’s next?

Tooling API Examples

Create a class

Choose an HTTP method to perform on the REST API service URI below:

GET POST PUT PATCH DELETE HEAD Headers Reset Up

/services/data/v37.0/tooling/sobjects/ApexClass Execute

Request Body

```
{"Name":"MyNewToolingClass",
"Body":"public class MyNewToolingClass{}"}
```

Expand All | Collapse All | Show Raw Response

- id: **01po0000005AnmcAAC**
- success: **true**
- errors

Imagine if you will...

The company gets Leads from across the globe, and every country that it gets a lead from has an SLA (Service Level Agreement) to respond.

The SLAs change monthly based on some business rules. SLAs are one of these values:

One Day
Two Days
Three Days
Five Days

Based on the SLA of the Country the Lead gets assigned to the appropriate Queue.

Singleton

What is it?

When you want there to be exactly one instance of a class, and it must be accessible to other clients over the life of a transaction

When to use them?

Iterating over a collection and instantiating a class that doesn't change across iterations

Needing the same instantiated class in a before and after trigger that doesn't change across the trigger executions

If you have related triggers in an execution that executes the same code that doesn't change

Singleton Implementation

Singleton, continued

What makes it work?

Hides the constructor by making it private.

Have a a class operation - typically a static function getInstance()

getInstance() - lazy loads the instance and guarantees only one instance is created. and ensures that a singleton is created and initialized only once

UML

Notes

- As developers, we are all familiar with the concept of DRY, but what's the most effective way to achieve this? Join us to explore design patterns for writing code that removes metadata dependencies and conforms to agile development. Structuring your code this way reduces the amount of repetitive code and test methods you'll write, while making it more maintainable and reusable. We'll highlight techniques using Dynamic Apex and the Tooling API that can be used across multiple code bases so you can create code that consumes fewer governor limits. We'll also cover techniques for addressing optional Salesforce features that help in developing managed packages.

Notes from Dan

- Tooling api to do templating- make somc code that figures out what i'm doing
- show how to use the tooling api to write code, and then
- using static apex/ soql - we have this problem.
- we refatctor it with dynamics showing code re-use
- siilar problem using classes/ objects. can't solve it no reflection, no late binding...but with tooling api you can do that. with the correct param types and types.

get rid of theoretical CS stuff.

- talk about them

Problem statement

involves records and fields

involves multiple fields on an object and i'm doing the same thing. I'm doing lead routing, and maybe i have a custom setting that decides what to do.

soo…… have a context calling a util class….. move that util class to stratatagies

Say case and lead routing. use a custom setting - if this industry and it goes to this person.

Usually with custom settings you want to put it in a wrapper.

After you demo it- use some language.

What does an ideal Generic Solution look like?

Ideals

- SOLID (single responsibility, open-closed, Liskov substitution, interface segregation and dependency inversion
- GRASP General responsibility assignment software patterns
- DRY - don't repeat yourself

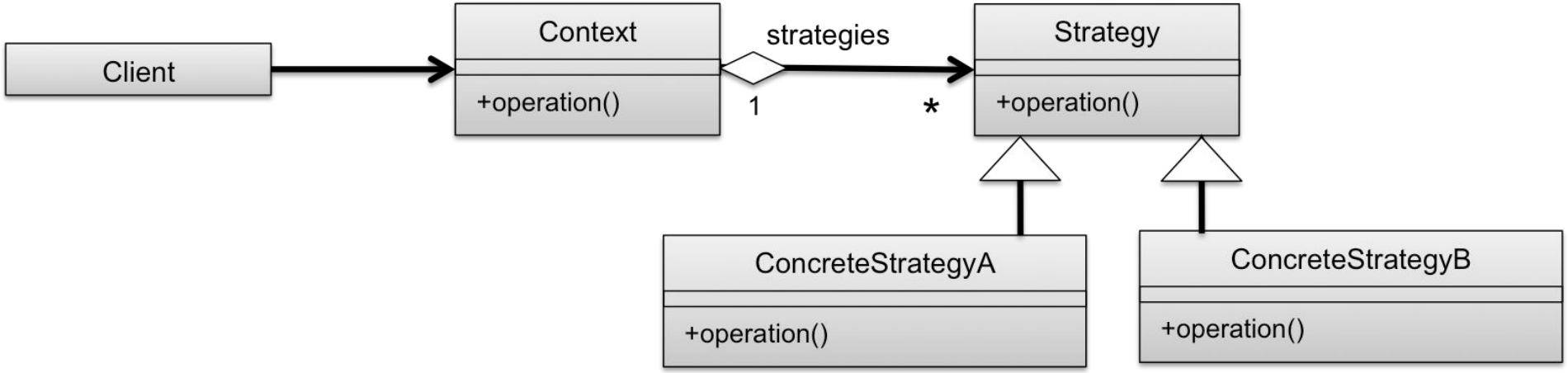
Limitations

No Generics in Apex

Design Pattern- Strategy, continued

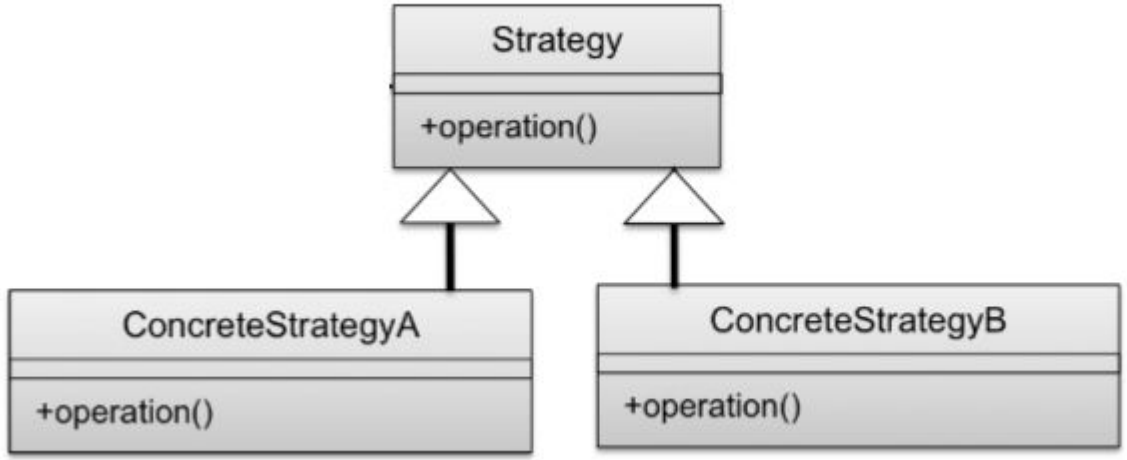
What makes it work?

- Lots of OOP
- Clear roles of the participants
- Communication mechanism between classes



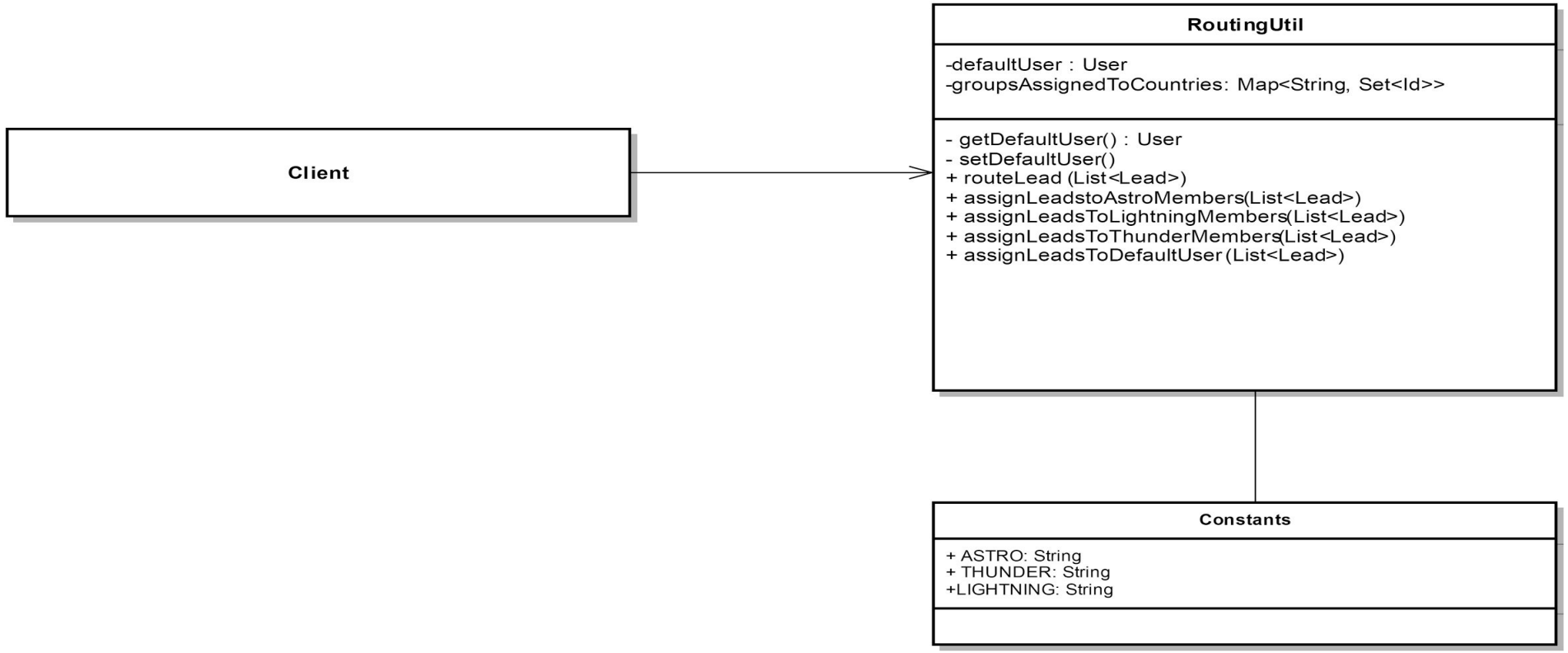
Design Pattern- Strategy, continued

Strategy



Design Pattern- Strategy, continued

Client, Context, and Strategy


Client
RoutingUtil
-defaultUser : User
-groupsAssignedToCountries: Map<String, Set<Id>>
- getDefaultUser() : User
- setDefaultUser()
+ routeLead (List<Lead>)
+ assignLeadstoAstroMembers(List<Lead>)
+ assignLeadsToLightningMembers(List<Lead>)
+ assignLeadsToThunderMembers(List<Lead>)
+ assignLeadsToDefaultUser(List<Lead>)
Constants
+ ASTRO: String
+ THUNDER: String
+LIGHTNING: String

Foundation

A tool to make your code
- SOLID
- GRASP
- DRY

Leverages OOP principles.
- Inheritance
- Abstraction
- Interfaces
- Polymorphism
- Encapsulation