# Crank

ArcMind Inc.

info@arc-mind.com

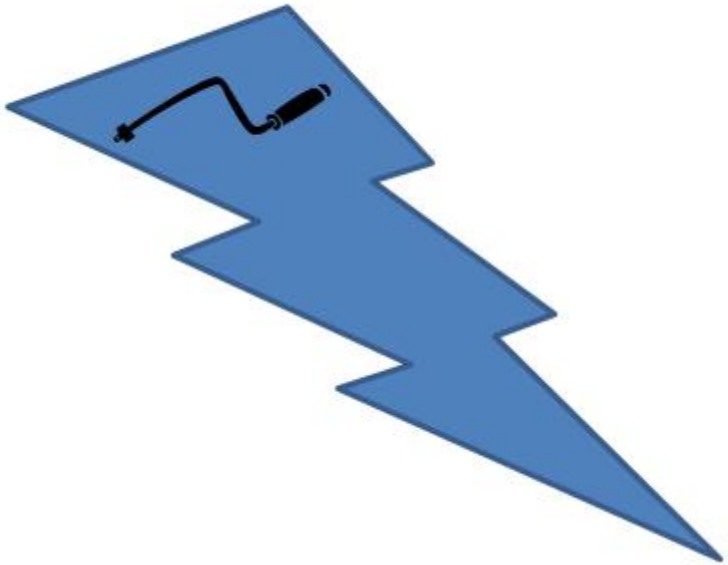
www.Arc-Mind.com

| | |
|---|---|
| Presenter: | Rick Hightower, CTO of ArcMind |
| Author: | Books Professional Struts, **Java Tools for Extreme Programming** (best seller), Struts Live (#1 download on TSS), etc.<br>JDJ Editorial Board, IBM DeveloperWorks regular author, JavaLobby Zone leader |
| Speaker: | ***JavaOne***, TheServerSide Symposium, JDJ Edge, SDWest, XP Universe |

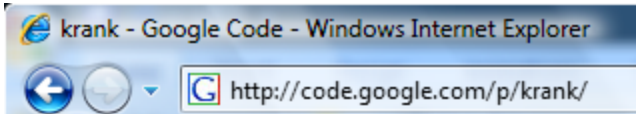
Rick is the founder of the Crank project

# Crank QuickStart Overview

- What is Crank?
- Understand how to use Crank



# Crank




Google Code

**krank**

*Java Framework for CRUD and Validation*

Crank is a master/detail, CRUD, and annotation driven validation framework built with JPA, JSF, Facelets and Ajax. It allows developers to quickly come up with JSF/Ajax based CRUD listings and Master/Detail forms from their JPA annotated Java objects.

Crank uses a lot of the new JSF features from Facelets, Ajax4JSF, etc. that will be used in JSF 2.0. Crank is a use case analysis of what is possible with the new JSF 2.0 stack.

The validation piece does server-side validation, Ajax validation or just emitted JavaScript? validation based on Java annotations, property files, XML files, or database tables. Currently works with JSF, Spring MVC and Spring Webflow.

The framework is named Crank as in: "crank out, to make or produce in a mass-production, effortless, or mechanical way: She's able to crank out one (CRUD listing) after another" and "crank up: to get started or ready", "to stimulate, activate, or produce", and most importantly "to increase one's efforts, output, etc.: Industry began to crank up after the new (CRUD framework became our corporate standard)." http://www.dictionary.com

# Crank Members

**License:** Apache License 2.0

**Labels:** JSF, CRUD, Validation, SpringMVC, SpringWebflow, Ajax, JavaScript, Regex

**Project owners:**

RichardHightower, chrismathiaster, bill.dudney, scottfauerbach, defurd, paulhixson, taboraz, tcellucci

**Project members:**

geoffc, johnfryar, sergem, daniloa.bonilla, jasond4747, wbogaardt, michaelmorett, sundukovskiy, sameera.veturi, parkleydondon, prakash.vatlam, reggiediamond, cagatay.civici, igor.no.12, vicken, seanjburns, dlwhitehurst

We have members from all of the U.S and the world. One member is South Africa, another hails from Turkey

# Crank Introduction

- Crank: “Crank the project out”
- Idiomatic Java GUI development
  - POJO Based
- GUI support for JPA projects
- Allows easy creation of CRUD listings and Master Detail pages to edit JPA based objects
- Uses JPA/JavaBean constructs to develop GUI
- Currently supports JSF as GUI

- Additional Goodies:
  - Annotation Driven Finder method mix-ins for DAO,
  - Criteria DSL for DAO,
  - Annotation Driven Validation framework

# Why Crank?

- Use current, common technology stack: Spring, Spring MVC or JSF, and JPA to implement CRUD GUIs quickly

- CRUD = Create Read Update and Delete
- CRUD has been done before
- CRUD is not the interesting part of the application
- Make CRUD easier
- Allows developers to focus on interesting parts of application

- Use annotations from JPA etc. to help build GUI
  - Map datatypes Date
  - Map relationships to GUI elements

# Crank has many parts

- **core**
  - Core crank support
- **validation**
  - Annotation/meta-data driven validation framework with Ajax bindings
  - Not tied to JSF or Spring MVC
- **crud**
  - Support event driven CRUD development, paginators, selectMany support, etc.
  - Not tied to JSF or Spring MVC
- **jsf-support**
  - Bind crank-crud to JSF
- **jsf-validation**
  - Bind crank-validation to JSF
- **springmvc-validation**
  - Bind crank-validation to Spring mvc
- **springmvc-crud**
  - planned
- **test-support**
  - Support for testing in JSF, Spring, Crank environment

# Crank examples

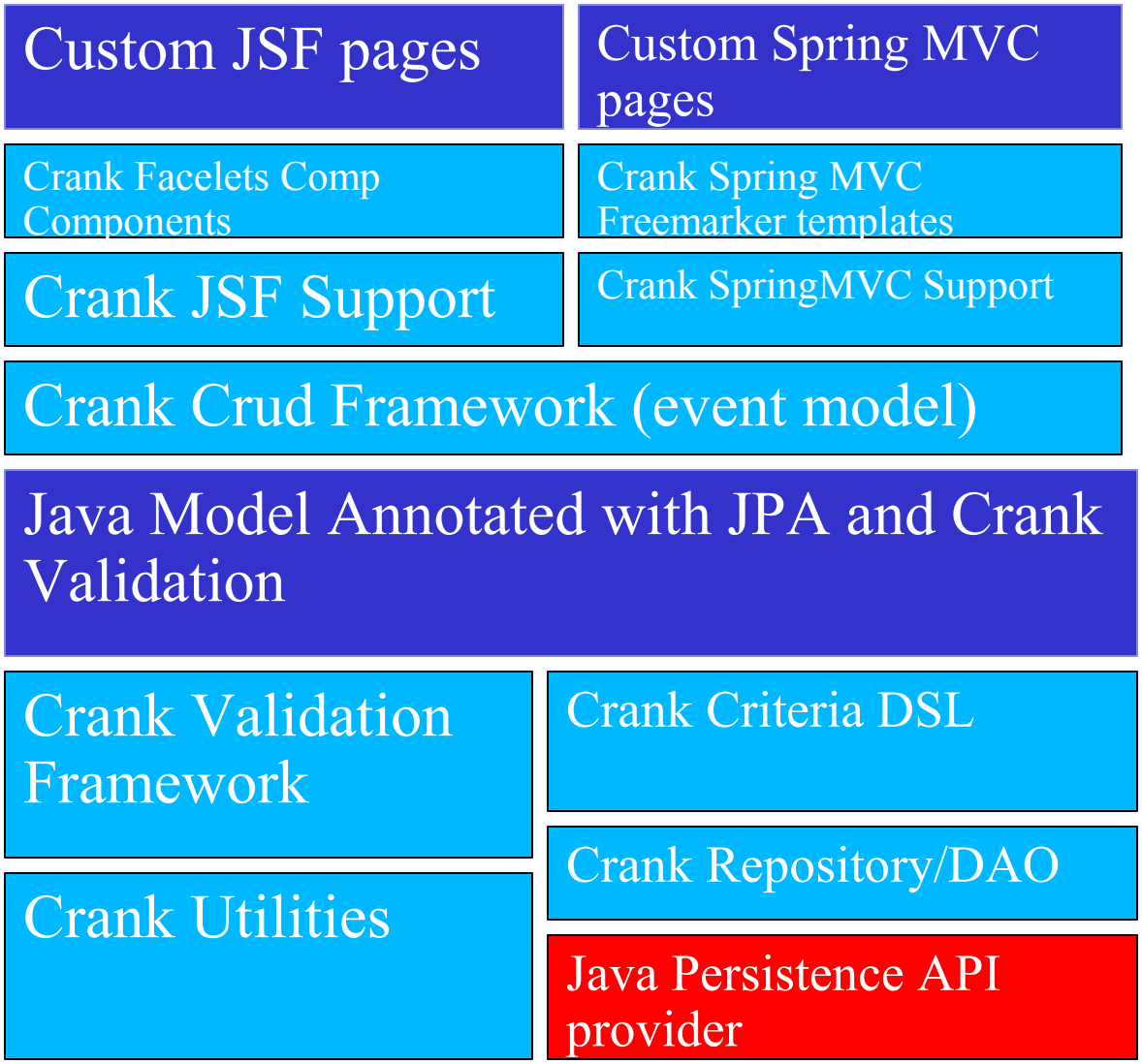
- **blank-project**
  - Blank starter project for creating new crank projects
- **crank-crud-webapp**
  - Project to show all of crank features (also project we use to test new features)
- **crank-validation-jsf-webapp**
  - Project to show how to use crank validation with JSF
- **crank-validation-springmvc-webapp**
  - Project to show how to use crank validation with Spring MVC
- **todo-example**
  - Simplest crank crud example that implements a simple TODO list

# Using Crank

- Define Persistence Tier in Java/JPA
- Use Facelets/JSF composition components to map persistence tier to GUI

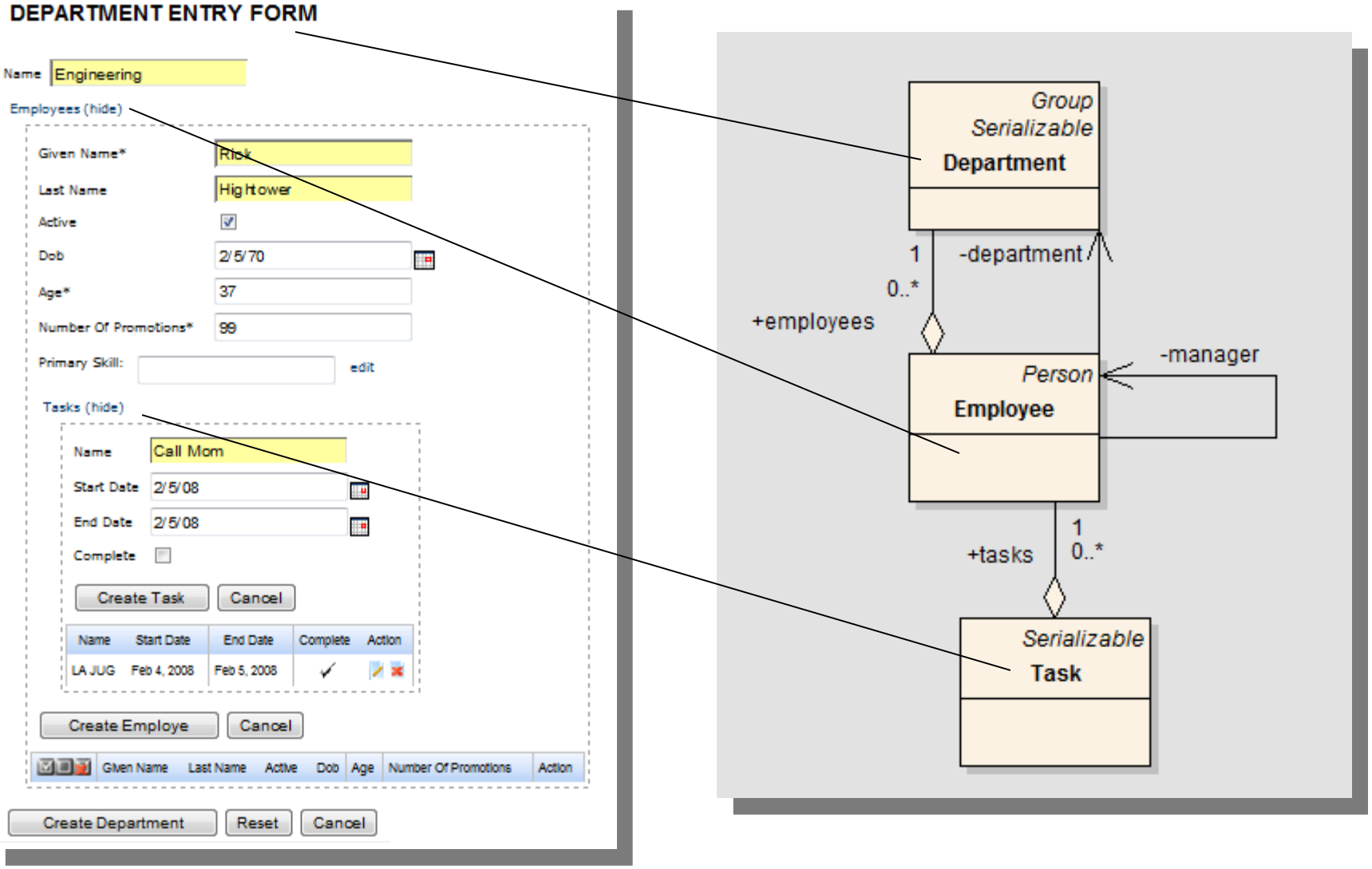
# Overview of Crank Features and Architecture

# Crank Block Diagram



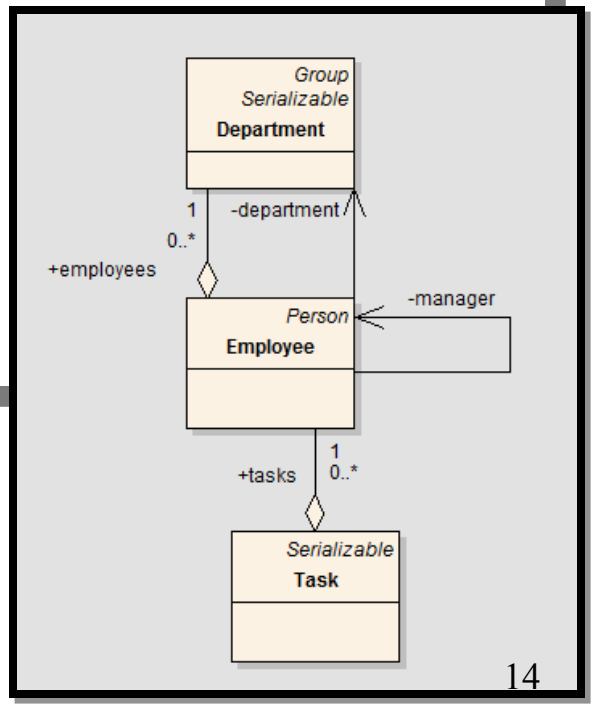
# Feature Overview

# Master Detail Framework

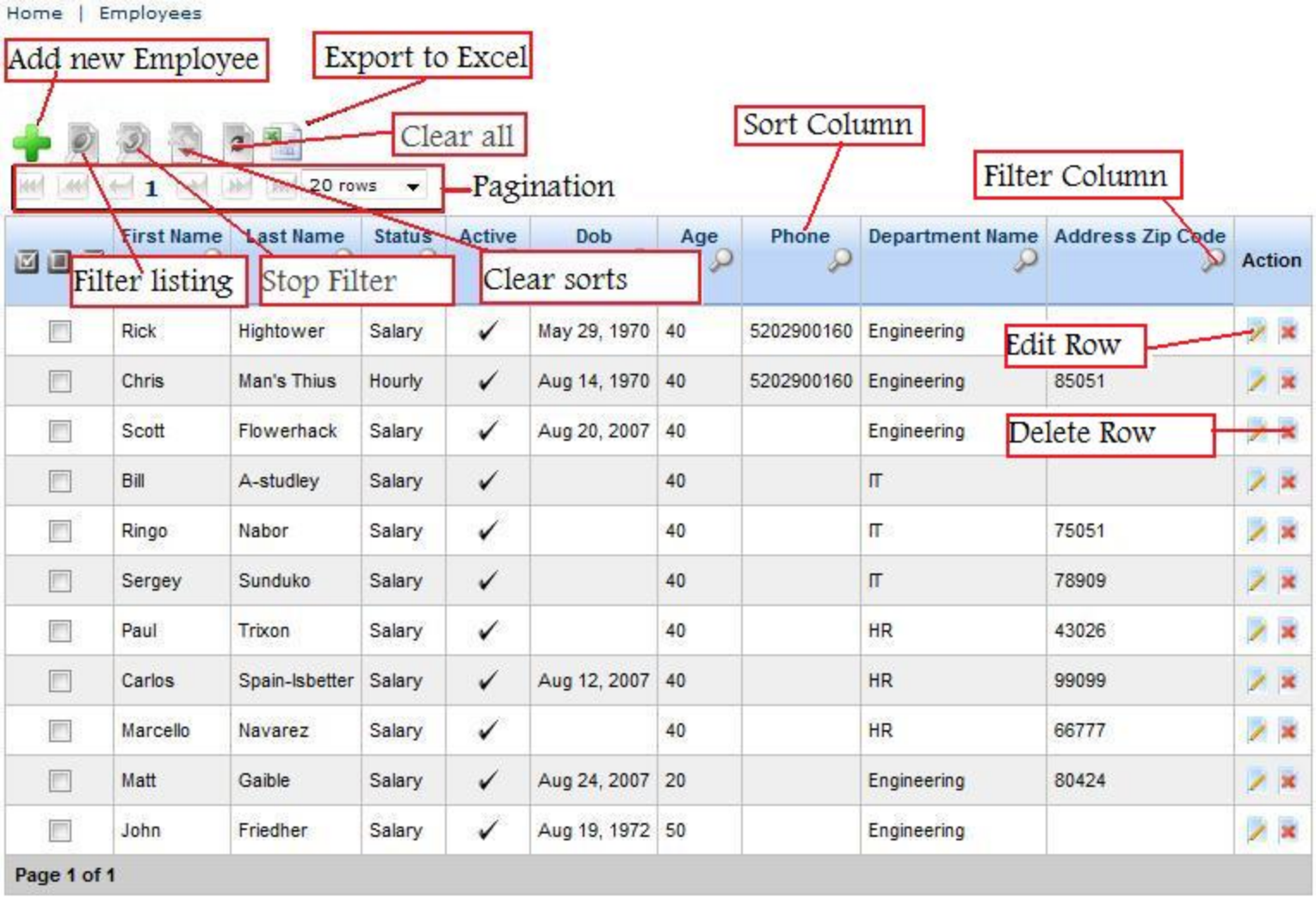


# Master Detail Framework

```
<crank:form crud="${crud}" propertyNames="name">
    <crank:detailListing
        detailController="${employeeDetailController}"
        propertyNames="firstName,lastName,active,dob,age,numberOfPromotions"
        parentForm="departmentForm">

        <crank:selectOneListing jsfSelectOneController="${employeeToSkillController}"
            propertyNames="name"
            parentForm="departmentForm"
            />

        <crank:detailListing
            detailController="${taskDetailController}"
            propertyNames="name,startDate,endDate,complete"
            parentForm="departmentForm"
            />

    </crank:detailListing>
</crank:form>
```



# Crud Listing (all columns are filterable and sortable)



# Listing Filters

Strings, Enums, Dates, Number, Related properties (employe.department.name)

**EMPLOYEE LISTING**

10 rows

| | Given Name filter | Surname filter | Status filter | Active | Dob filter | Age | Phone | Department Name filter | Address Zip Code | Specialty Name | Action |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | R | High | Hourly | | 2/4/69 thru 2/12/74 | | | Eng | | | |
| | Rick | Hightower | Hourly | | Feb 4, 1970 | 20 | 5202900160 | Engineering | 85748 | | |

```
<crank:listing  jsfCrudAdapter="${employeeCrud}"
    propertyNames="firstName,lastName,status,active,dob,age,phone,department.name,address.zipCode,specialty.name"
    parentForm="employeeListingForm"
    />
```

# Editing Relationships--Select One Controller

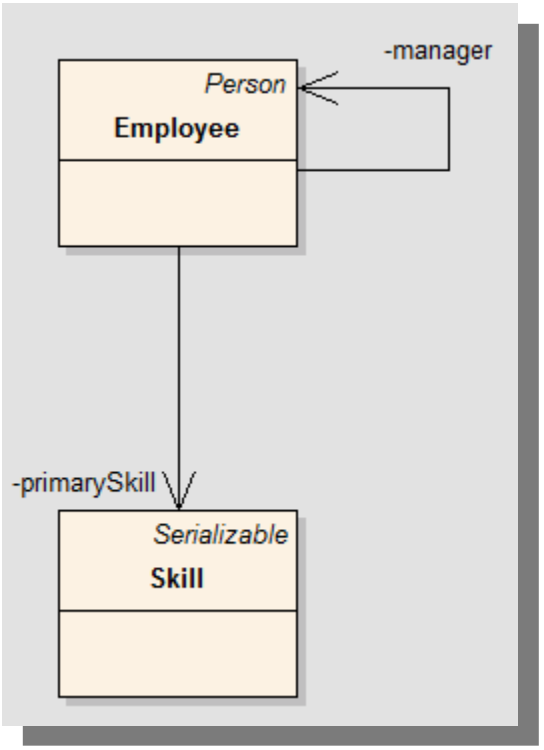
Primary Skill: sill2 edit

Primary Skill: sill2 (cancel)

1 10 rows

| | Name |
|---|---|
| select | sill2 |
| select | skill1 |
| select | skill3 |

Page 1 of 1




```
<crank:selectOneListing jsfSelectOneController="${employeeToSkillController}"
    propertyNames="name" parentForm="employeeForm" />
```

# Editing Relationships -- Select Many Controller

Roles: role3, role1, role2 ...

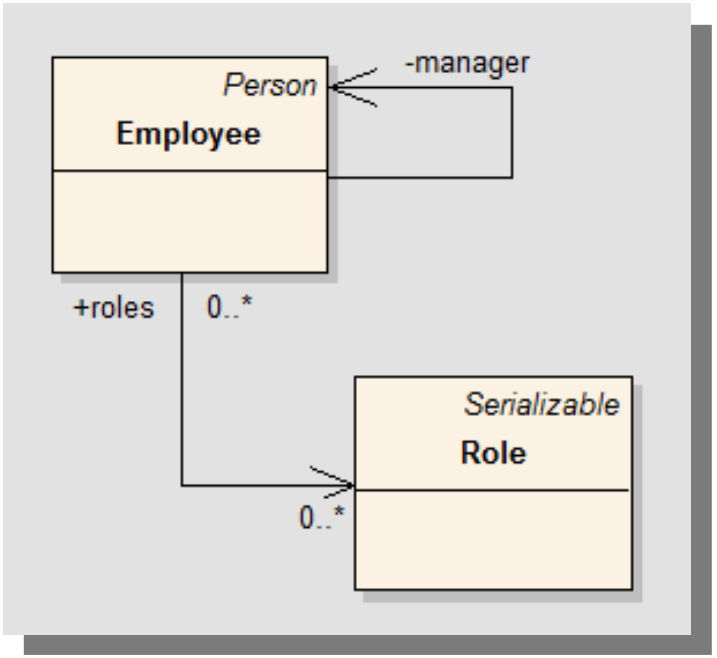
Roles: role3, role1, role2

1 10 rows

| | Name |
|---|---|
| ☑ | role1 |
| ☑ | role2 |
| ☑ | role3 |

Page 1 of 1

Update Cancel




```
<crank:selectMany jsfSelectManyController="${employeeToRoleController}"
    propertyNames="name" parentForm="employeeForm" />
```

# Crud framework builds on top of Criteria API

- JPA DAOs process Criteria API
- Criteria API just POJOs
  - you can write your own classes that read them
  - Not tied to JPA; Examples that use SQL to process Criteria API

```
employees = employeeDao.find(or(eq("department.name", "Engineering"),
        like("firstName", "Ri")));
```

```
List<Employee> employees = employeeDao.find(in("age", 1, 2, 3, 4, 5, 6));
```

```
List<Employee> employees = employeeDao.find(between("age", 1, 100));
```

```
List<Employee> result = employeeDao.find(join(joinFetch("o.department", true, "foo")),
        eq("foo.name", true, "Engineering"));
```

```
personDao.find(join(entityJoin("Employee", "e")),
        eq("e.firstName", true, "Rick"),
        eq("ssn", "333333311"),
        objectEq("o","e"));
```

# Annotation driven validation

- Annotations are read and processed

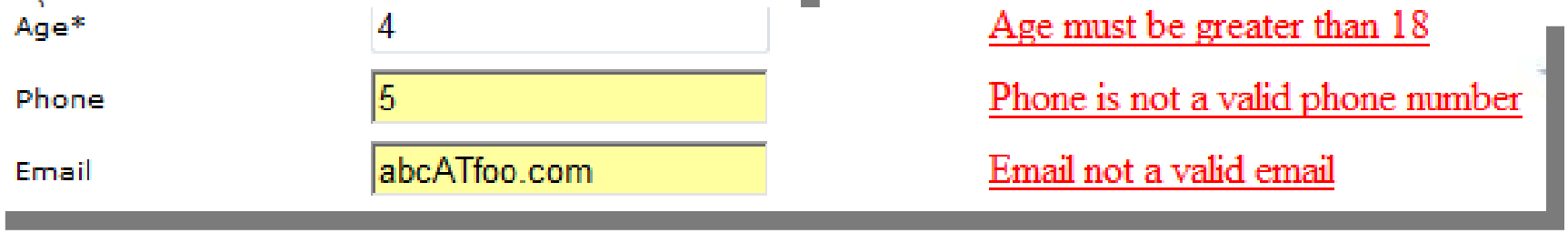
Employee.java

```
@Email
public void setEmail( String email ) {
    this.email = email;
}

@Required
@LongRange( min = 18L, max = 135L )
public void setAge( int age ) {
    this.age = age;
}

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

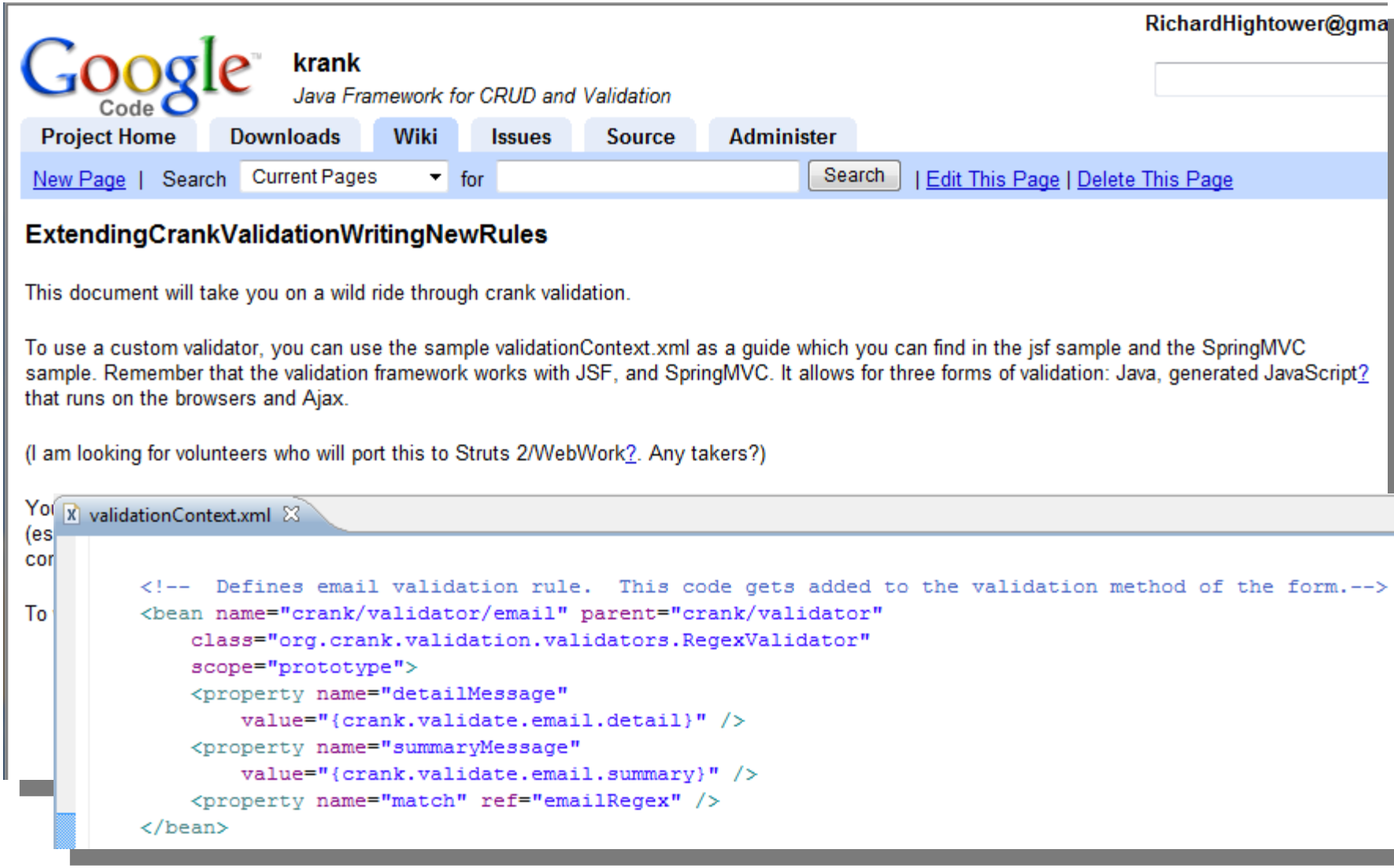
# Validation used by GUI layer

Employee.java

```
@Email
public void setEmail( String email ) {
    this.email = email;
}

@Required
@LongRange( min = 18L, max = 135L )
public void setAge( int age ) {
    this.age = age;
}

@Phone
public void setPhone( String phone ) {
    this.phone = phone;
```



# You can customize validation (configured in Spring)



# Small History of Crank

# Evolution of Crank

- 1st Web Crud Framework I wrote written with Struts, EJB 2.x and JSP (2001)
- Had validation based on property names
- Used Jython to generate starter code
  - Learned about tradeoffs between code gen vs. framework
- 2nd Web Crud Framework JSF, 2004
  - Wow, nice and easy; needed many CRUD listings wrote small framework
- 3rd Web Crud Framework Presto, 2005
- 4th Web Crud Framework Crank

# Presto (Crank's closest cousin)

- Internal project, started as an example on how to combine JSF, Spring and Hibernate (Java 1.4 no annotations; big company)
  - 18 months (3 day contract)
- Kept adding features, plan was to OpenSource
- Added code generator (Maven 2 plugins); heavy use of Facelets
- Now has fulltime support staff, writes documentation and FAQ
- Corporate Standard for new Apps (large corporation)
- 20 to 30 applications launched on Presto, many more expected
- Very mature, lots of documentation, etc.
- Light on Java 5 features;
- Ported to work with iBatis in addition to JPA
- Crankification of Presto (possibly)

# Crank

- Open Source… Presto's cousin (a rewrite)
- Developed while working on applications for Vantage Media
  - Vantage Media looking for senior Java developers, and is a great place to work
- Vantage Media, ArcMind Inc., corporate sponsors
- Crank makes heavy use of Annotations, and has support for Enums, etc.
- Crank is Ajaxified via Ajax4JSF
- Crank makes heavy use of Facelets
- Added Criteria API to base filtering and sorting on
- Controllers are divorced from JSF and are easier to test
- Not as mature as Presto, much more nascent
- No more reinventing the wheel

# Short term Roadmap (May 2008)

- Porting CRUD framework to Spring MVC (started)
  - Spin-off project started Crank on Struts (Struts 2)
- Writing Code generator for reverse engineering database tables into Crank listings (started)
- More integration with Seam (done at some level)
  - As an option, already started and already working
- More documentation
- Higher code coverage (started)
- Website, Public JIRA, Public Confluence (started process)
  - Currently hosted on google code, WIKI, etc.
- Crank 1.0 May 2008

# Longer Term Roadmap (Late 2008 and beyond)

- Port to JSF 2.0 when it comes out
- Port Facelets Composition components to other component frameworks (current uses Ajax4JSF, RichFaces and Tomahawk)
  - IceFaces
  - Trinidad
- Improve JPA handling via Apache Orchestra
- Port to other frontends, Swing, GWT, Flex?
- Really driven by needs of members

# Getting Started with Crank Worked Example

# Getting started with Crank (1 of 2)

- Check out crank project
- Build Crank
- Copy the ***blank-project*** from examples

  Use Beyond Compare or Araxis (don't copy .svn folders)
- Edit ***pom.xml*** file and rename project
- Create Eclipse projects or IntelliJ, import project into IDE

# Getting started with Crank (2 of 2)

- Refactor/Rename ***CrankCrudExampleApplicationContext***
- Create JPA enabled ***Task*** class
- Add entry in new App context for ***Task***
- Add persistence.xml entry for ***Task*** persistence object
- Add form and listing to ***home.xhtml***

# Check out crank project

- **Using subversion, you need to check out crank**
- We created a tag called ***good***, which is the last known ***good*** build
- Create a new directory, in the new directory run the following command from the command line:

```
svn checkout http://krank.googlecode.com/svn/tags/good crank
```

# Build Crank

- Crank uses maven for builds
- Before you can use an example Crank project you need the Crank jar files
- The easiest way to get the Crank jar files is to build Crank as follows from the command line:

```
cd crank
mvn clean install -Dmaven.test.skip=true
```

# Copy the blank-project from examples

- Copy the ***blank-project*** from examples
- **Crank ships with a blank-project**
- **The blank-project is everything you need to create your first crank project**
- **JSF, Facelets, Ajax4JSF, Spring and JPA are preconfigured**
- **Copy examples\blank-project (in the crank directory) to a new location**

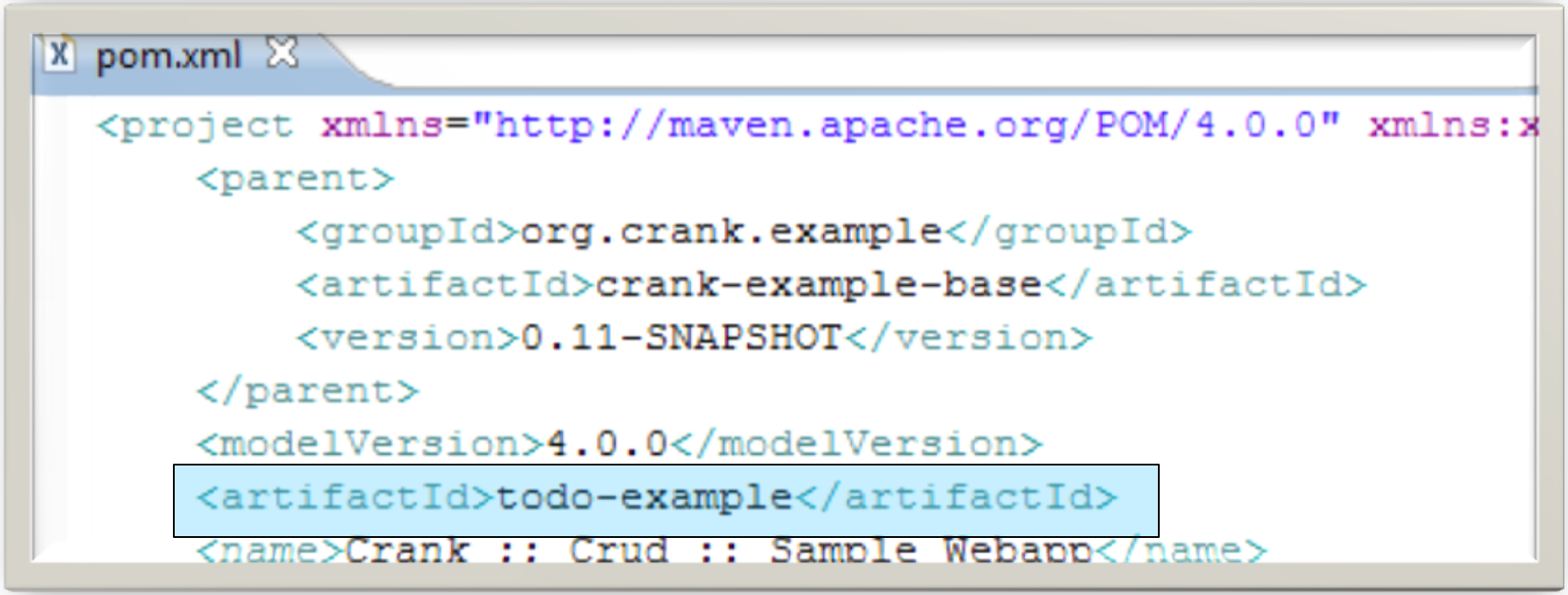
# Edit pom.xml file and rename project (1 of 2)

- In the new folder, edit the pom.xml file
- Change the artifact-id from blank-project to todo-example
- <artifactId>todo-example</artifactId>

- DO NOT MODIFY THE PARENT artifactId

# Edit pom.xml file and rename project (2 of 2)

- Edit the artifact Id



# Create Eclipse projects or IntelliJ

- IDE support
- To create Eclipse project files run:

```
mvn eclipse:eclipse
```

- To create InteliJ project files run:

```
mvn idea:idea
```

- Now import the project into your favorite IDE

# Using IDE rename CrankCrudExampleApplicationContext (1 of 2)

- Using IDE rename/refactor CrankCrudExampleApplicationContext to org.crank.todo.TodoApplicationContext
- Make sure IDE scans xml files when refactoring
- Ensure name changes in \src\main\resources\applicationContext.xml

# Using IDE rename CrankCrudExampleApplicationContext (2 of 2)

- *Ensure change in \src\main\resources\applicationContext.xml as follows:*

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/sche
           http://www.springframework.org/schema/aop http:/

    <bean class="org.crank.todo.TodoApplicationContext" />
```

# Create JPA enabled Task class

Task.java

```
package org.crank.todo;

import java.io.Serializable;

@Table (name="TODO")
@Entity
public class Task implements Serializable {
    @Id @GeneratedValue (strategy=GenerationType.AUTO)
    private Long id;
    @Column (length=40)
    private String name;
    @Column (length=256)
    private String description;

    private boolean complete;
```

# Add entry in new App context for Task

Task.java | TodoApplicationContext.java

```
package org.crank.todo;

import java.util.ArrayList;

@Configuration(defaultLazy = Lazy.TRUE)
public abstract class TodoApplicationContext extends CrudJSFConfig {

    private static List<CrudManagedObject> managedObjects;

    @Bean(scope = DefaultScopes.SINGLETON)
    public List<CrudManagedObject> managedObjects() {
        if (managedObjects == null) {
            managedObjects = new ArrayList<CrudManagedObject>();
            managedObjects.add(new CrudManagedObject(Task.class, null));
        }
        return managedObjects;
    }
```

# Add persistence.xml entry for Task persistence object

- src/resources/META-INF/persistence.xml

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/
    xmlns:xsi="http://www.w3.org/2001/XMLSch
    xsi:schemaLocation="http://java.sun.com/
            http://java.sun.com/xml/ns/persi
    version="1.0">

    <persistence-unit name="blank-project"
        transaction-type="RESOURCE_LOCAL">
        <class>org.crank.todo.Task</class>
    </persistence-unit>
```

# Add form and listing to home.xhtml

home.xhtml

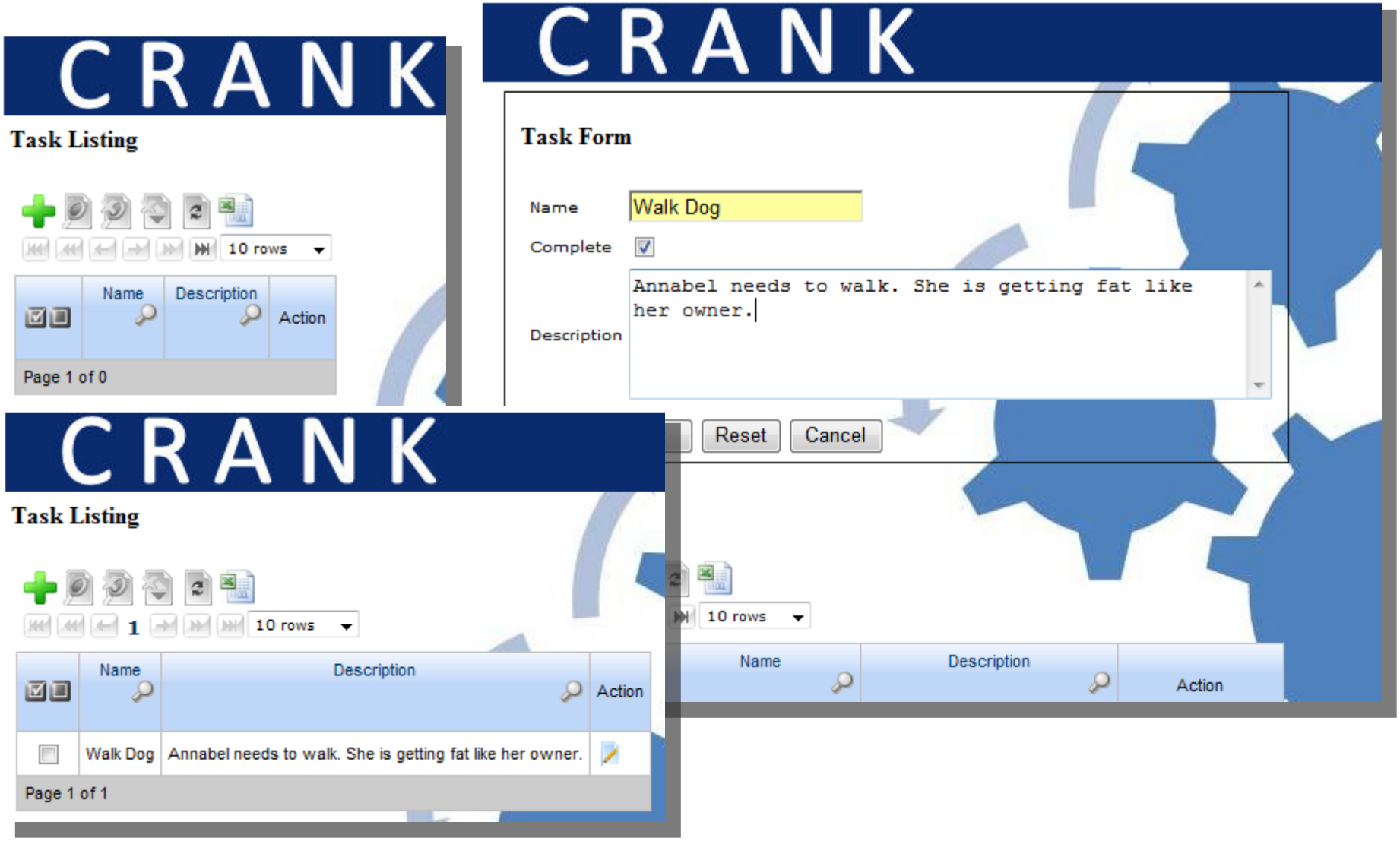
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:crank="http://www.googlecode.com/crank">

<ui:composition template="/templates/layout.xhtml">
    <ui:define name="content">
        <h:form id="taskForm">
            <crank:ajaxForm crud="${cruds.task.controller}"
                parentForm="taskForm"
                propertyNames="name,complete,description" />
            <h4>${cruds.task.controller.name} Listing</h4>
            <crank:listing jsfCrudAdapter="${cruds.task}"
                propertyNames="name,complete,description"
                parentForm="taskForm" />
```

# Running the example

- To run the example, run mvn jetty:run from the command line:

```
mvn jetty:run
```

# Using Crank Todo App

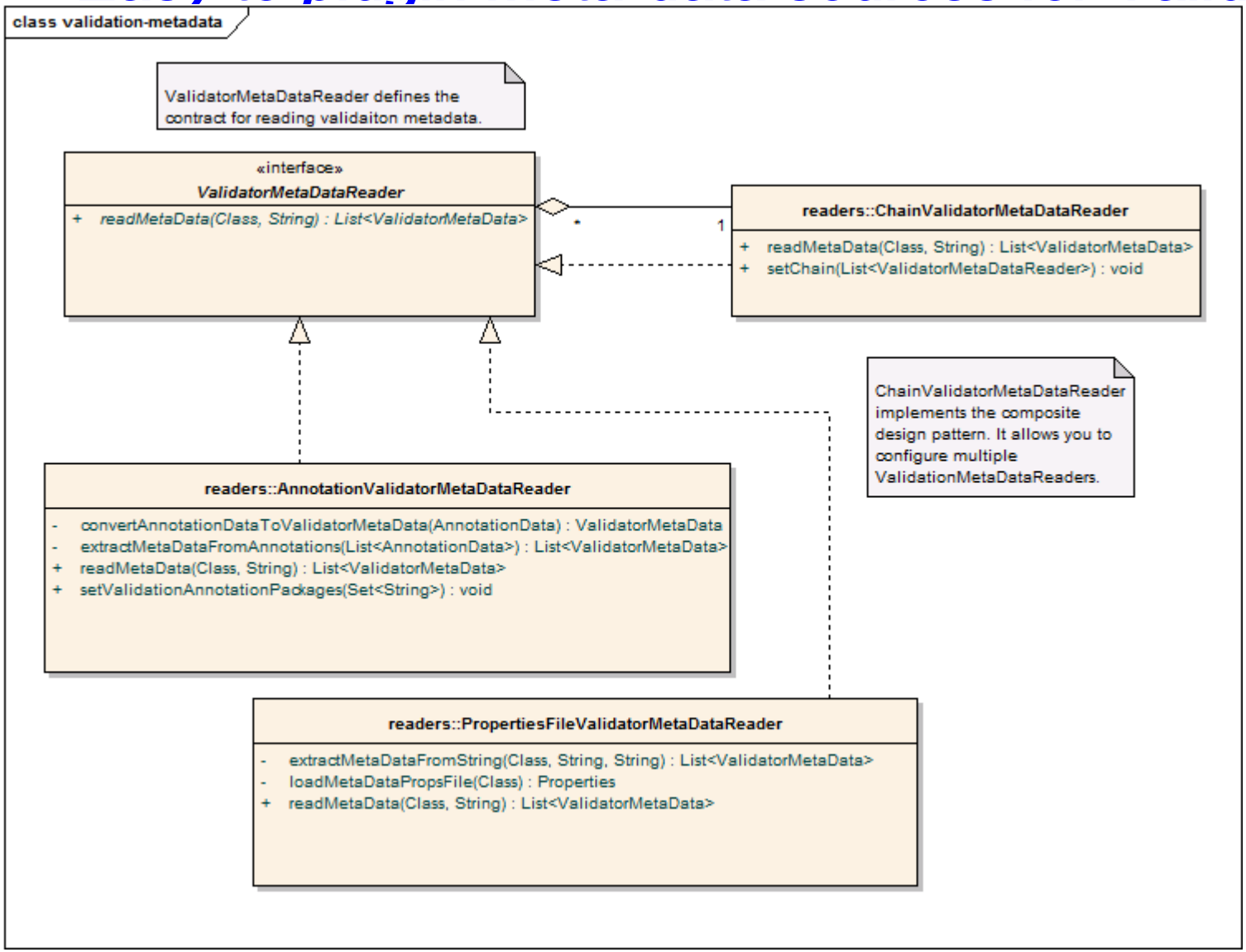


# Where to go from here

- **Crank home page**
  - http://code.google.com/p/krank/
- **Crank Crud Design Document**
  - http://code.google.com/p/krank/wiki/CrankCrudDesignDocs
  - If the instructor has time, go through the design document
- **Crank Crud Intro**
  - http://code.google.com/p/krank/wiki/CrankCrudIntro
- **More involved Crank Crud Tutorial**
  - http://code.google.com/p/krank/wiki/CrankCrudTutorial

# Validation Framework Design Overview

# Easy to plugin meta-data sources for Validation



# ValidatorMetaDataReader

- *ValidatorMetaDataReader*
- extention point for classes that need to read validation meta-data
- One implementation reads the meta-data from a properties file
- One implementation reads the data from Java 5 Annotation
- Another implementation reads from DB

# AnnotationValidatorMetaDataReader

- Reads validation meta-data from annotations.
- Configure base package of the annotatoins
  - defaults to "org.crank.annotations.validation".
- Takes name of the ValidatorMetaData and captilalizes the first letter.
- "com.mycompany.annotations", and ValidatorMetaData.name = "required", then look for com.mycompany.annotations.Required
- use annotation without polluting your model classes with Crank annotations
- Three extensions use this JavaScript, JSF and SpringMVC

# AnnotationValidatorMetaDataReader

**AnnotationValidatorMetaDataReader** reads validation meta-data from annotations.

This class reads a annotation as follows: You pass in the base package of the annotatoins it defaults to "org.crank.annotations.validation". It then takes the `name` of the `ValidatorMetaData` and captilalizes the first letter. Thus if you pass the package "com.mycompany.annotations", and `ValidatorMetaData.name = "required"`, then it will look for an annotation called com.mycompany.annotations.Required. The idea behind this is that you can use annotation without polluting your model classes with Crank annotations.

The parent class that owns the annotation should have annotation as follows:

```
Author:
      Rick Hightower
@Required
      @Length (min=10, max=100)
         public String getFirstName(){...
@Required
      @Range (min=10, max=100)
         public void setAge() {...
```

The **firstName** corresponds to a property of the Foo class. The **firstName** is associated with the validation rules **required** and **length**. The **length** validation rule states the minimum and maximum allowed number of characters with the **min** and **max** parameters.

Two different frameworks read this meta-data (curently). Our validation framework, which is mostly geared towards server-side validation and our client-side JavaScript framework, which is geared towards producing client-side JavaScript.

# PropertiesFileValidatorMetaDataReader

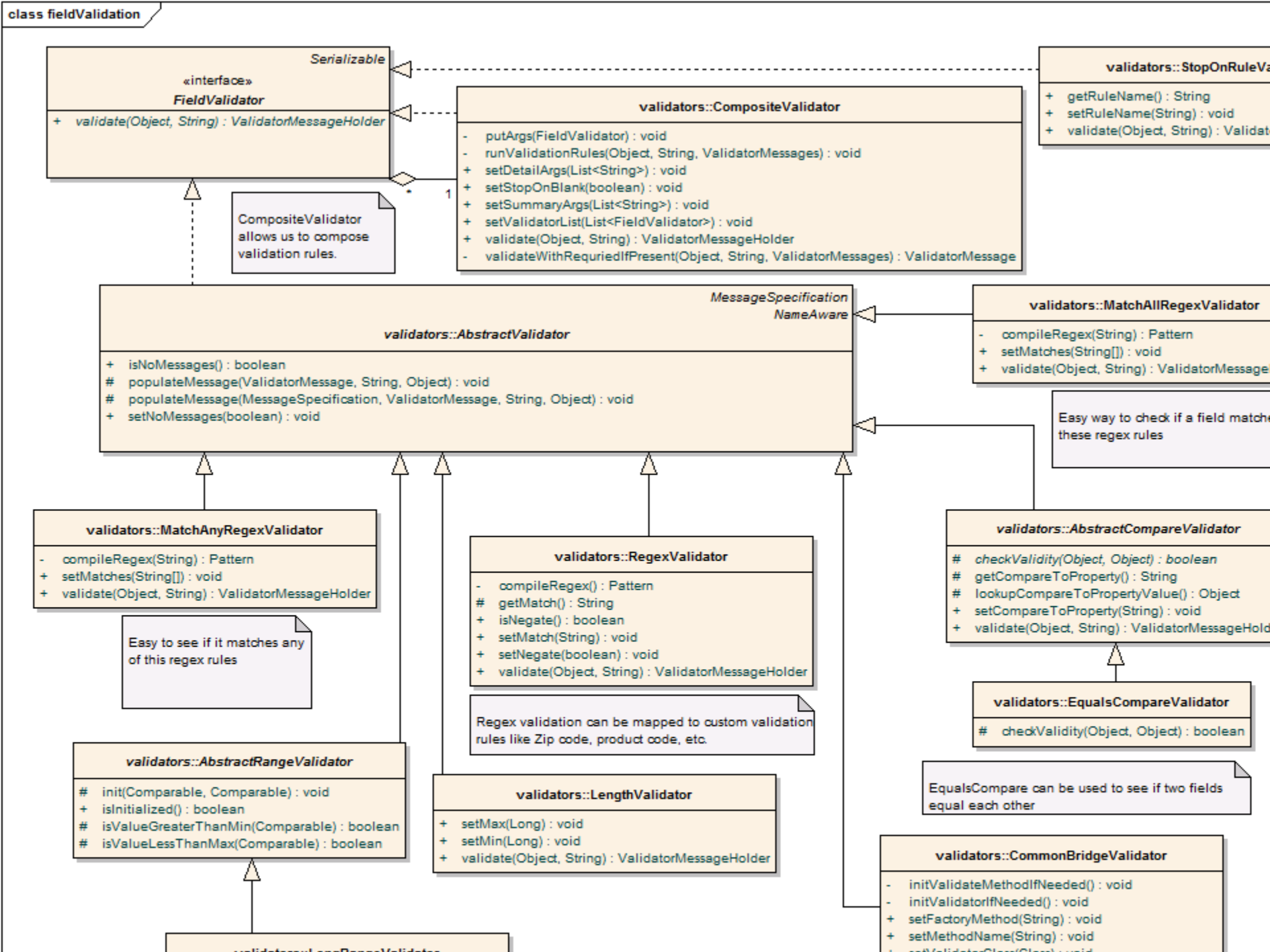
- reads validation meta-data from properties files
- Class name com.foo.Foo, then the resource name is com.foo.Foo.properties.
- Properties file contents

- firstName=required; length min=10, max=100
- age=required; range min=10, max=100

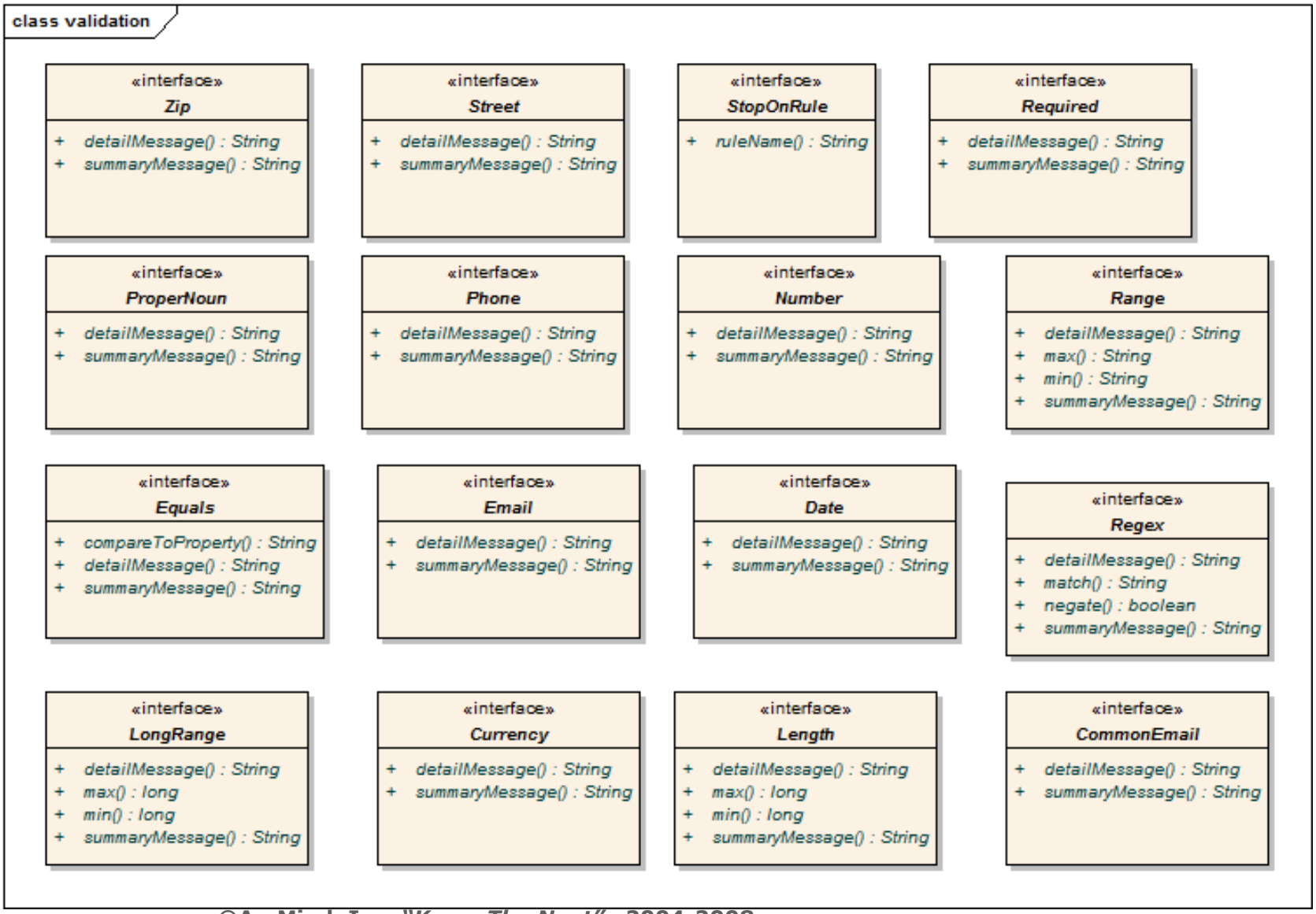
- First name is required, and must be at least 10 characters and no more than 100 characters
- Age is required and must be between 10 and 100

# ChainValidatorMetaDataReader

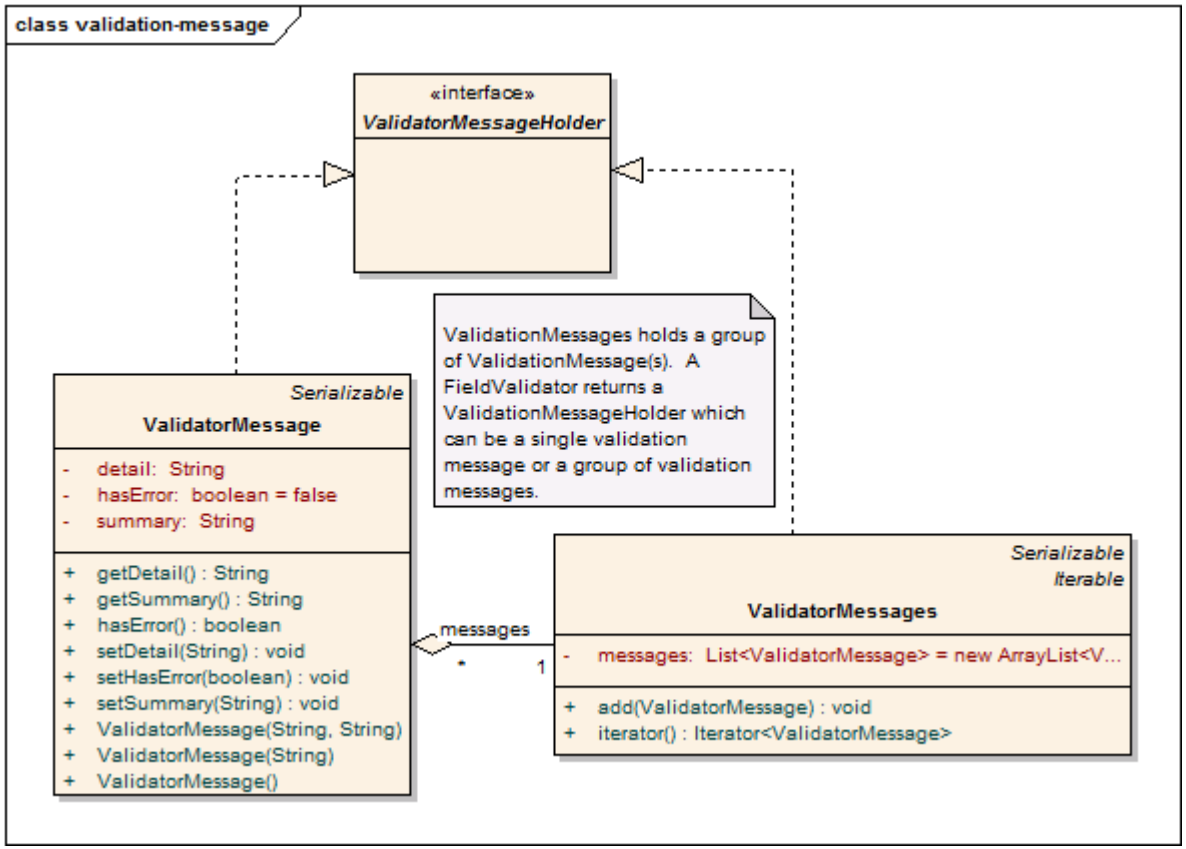
- Allows you to chain validator readers
- you can read validation data from more than one source
- you could read validation meta data from properties files, and annotations
- last one configured in the chain wins if you have the same type
- also merges as long as they are different types (there are overrides rules as well)


class fieldValidation
Serializable
«interface»
FieldValidator
+ validate(Object, String) : ValidatorMessageHolder
validators::StopOnRuleVa
+ getRuleName() : String
+ setRuleName(String) : void
+ validate(Object, String) : Validat
validators::CompositeValidator
- putArgs(FieldValidator) : void
- runValidationRules(Object, String, ValidatorMessages) : void
+ setDetailArgs(List<String>) : void
+ setStopOnBlank(boolean) : void
+ setSummaryArgs(List<String>) : void
+ setValidatorList(List<FieldValidator>) : void
+ validate(Object, String) : ValidatorMessageHolder
- validateWithRequriedIfPresent(Object, String, ValidatorMessages) : ValidatorMessage
*
1
CompositeValidator allows us to compose validation rules.
MessageSpecification
NameAware
validators::AbstractValidator
+ isNoMessages() : boolean
# populateMessage(ValidatorMessage, String, Object) : void
# populateMessage(MessageSpecification, ValidatorMessage, String, Object) : void
+ setNoMessages(boolean) : void
validators::MatchAllRegexValidator
- compileRegex(String) : Pattern
+ setMatches(String[]) : void
+ validate(Object, String) : ValidatorMessage
Easy way to check if a field matche
these regex rules
validators::MatchAnyRegexValidator
- compileRegex(String) : Pattern
+ setMatches(String[]) : void
+ validate(Object, String) : ValidatorMessageHolder
Easy to see if it matches any of this regex rules
validators::RegexValidator
- compileRegex() : Pattern
# getMatch() : String
+ isNegate() : boolean
+ setMatch(String) : void
+ setNegate(boolean) : void
+ validate(Object, String) : ValidatorMessageHolder
Regex validation can be mapped to custom validation rules like Zip code, product code, etc.
validators::AbstractCompareValidator
# checkValidity(Object, Object) : boolean
# getCompareToProperty() : String
# lookupCompareToPropertyValue() : Object
+ setCompareToProperty(String) : void
+ validate(Object, String) : ValidatorMessageHold
validators::EqualsCompareValidator
# checkValidity(Object, Object) : boolean
EqualsCompare can be used to see if two fields equal each other
validators::AbstractRangeValidator
# init(Comparable, Comparable) : void
+ isInitialized() : boolean
# isValueGreaterThanMin(Comparable) : boolean
# isValueLessThanMax(Comparable) : boolean
validators::LengthValidator
+ setMax(Long) : void
+ setMin(Long) : void
+ validate(Object, String) : ValidatorMessageHolder
validators::CommonBridgeValidator
- initValidateMethodIfNeeded() : void
- initValidatorIfNeeded() : void
+ setFactoryMethod(String) : void
+ setMethodName(String) : void

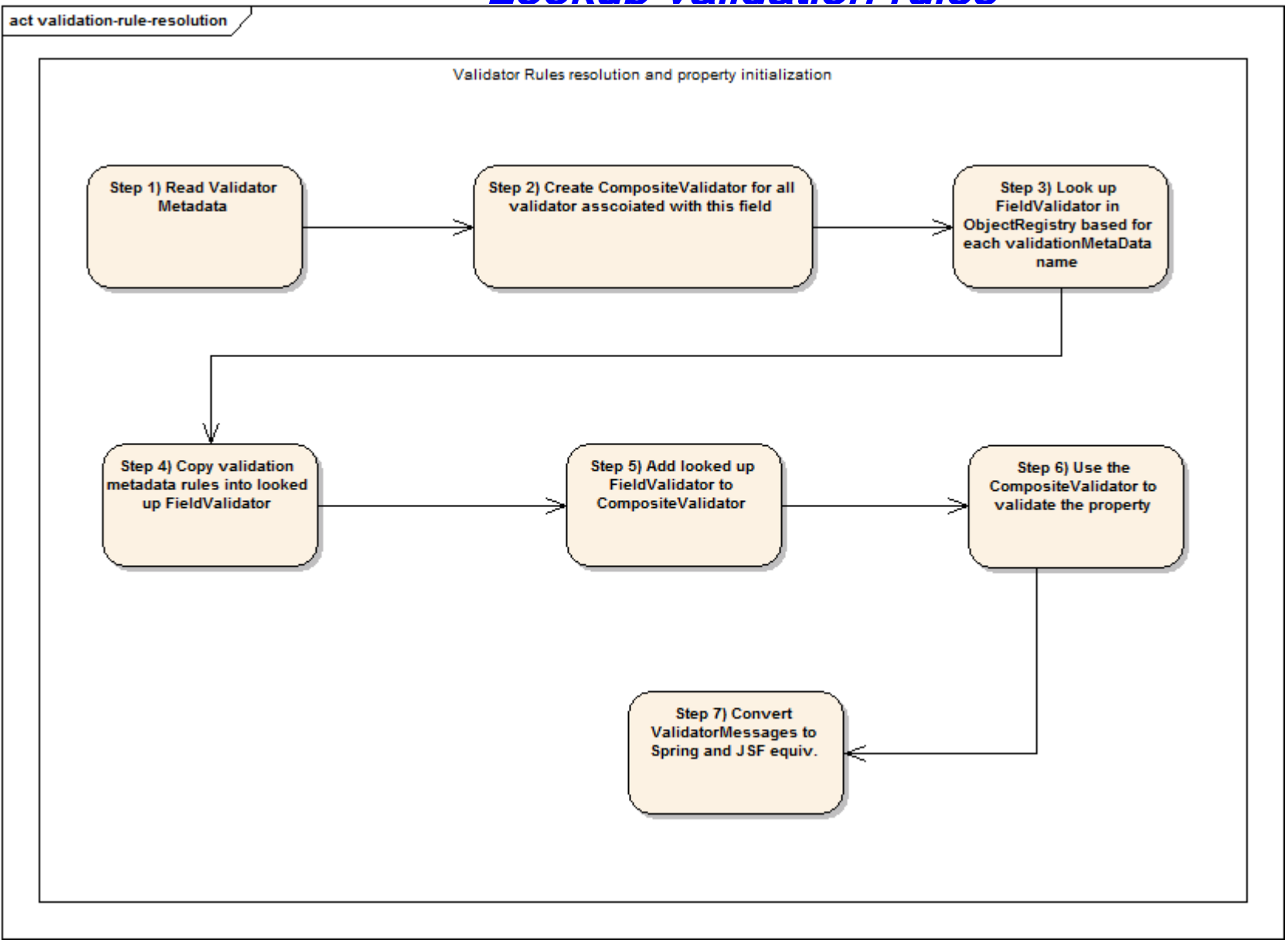
# Validation Annotations



# Validation Messages



# Lookup validation rules



# More on validation rules

- Design Document (8 pages)

http://code.google.com/p/krank/wiki/CrankValidationDesignDocument

- How to configure and extend

http://code.google.com/p/krank/wiki/ExtendingCrankValidationWritingNewRules

- Example using JSF and JavaScript

http://krank.googlecode.com/svn/trunk/examples/crank-validation-jsf-webap

- Example using Spring MVC and JavaScript

http://krank.googlecode.com/svn/trunk/examples/crank-validation-springmv