



StrataCode

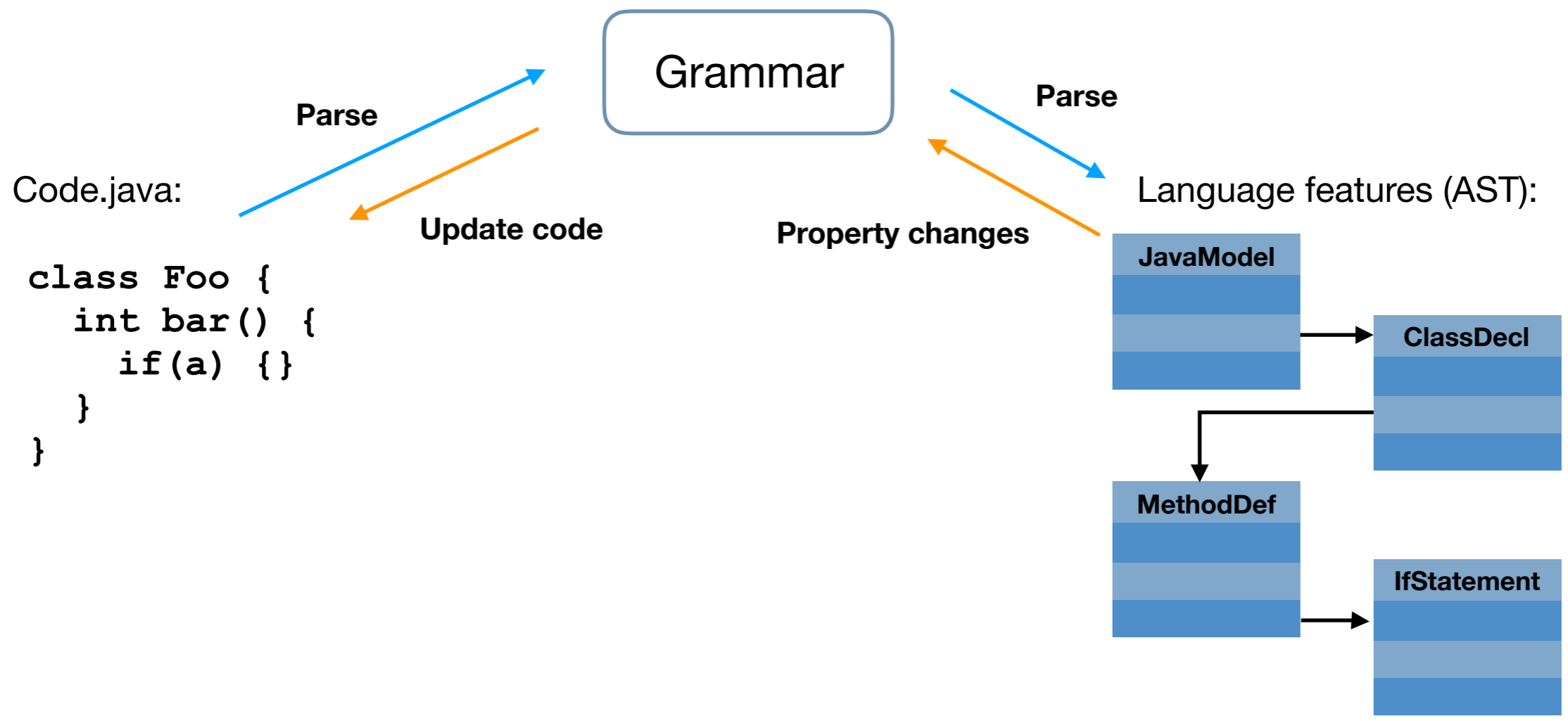
Do more than code - StrataCode



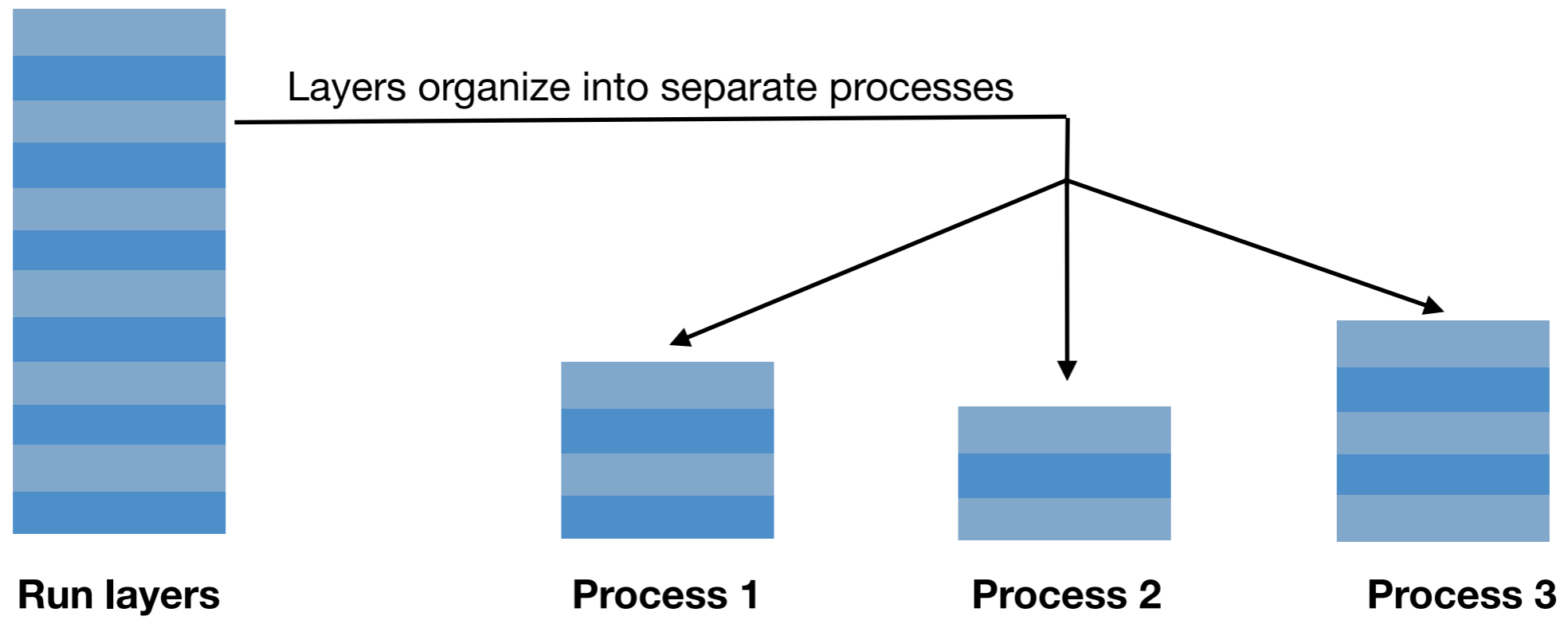
Introducing StrataCode

- Mostly solo project by Jeffrey Vroom
- Layers to organize code. Build declarative, efficient apps
- Software Architect: AVS, ATG, Adobe Flex Data Services
- Last ten years consulting and building StrataCode
- Looking for ideas to make it better, projects to build, and more partners to help
- Open source if there's enough support

Code processing framework



Multi-process build/run from one layer stack

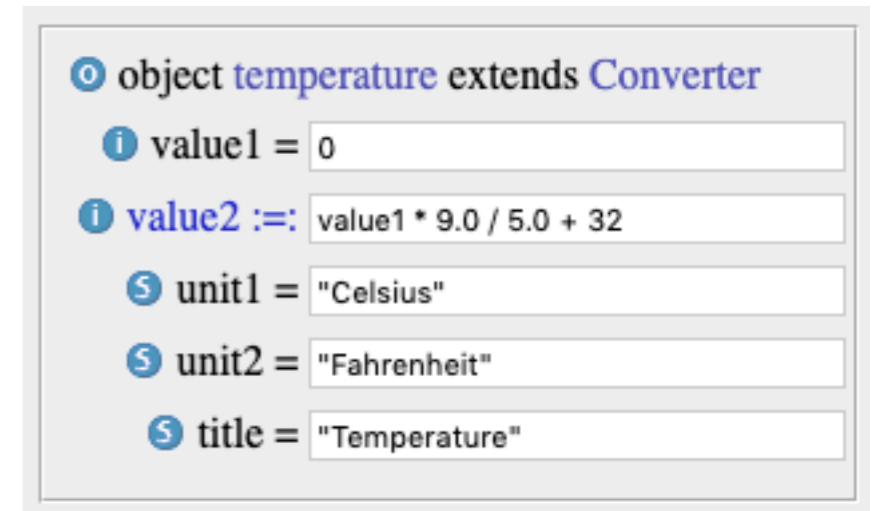
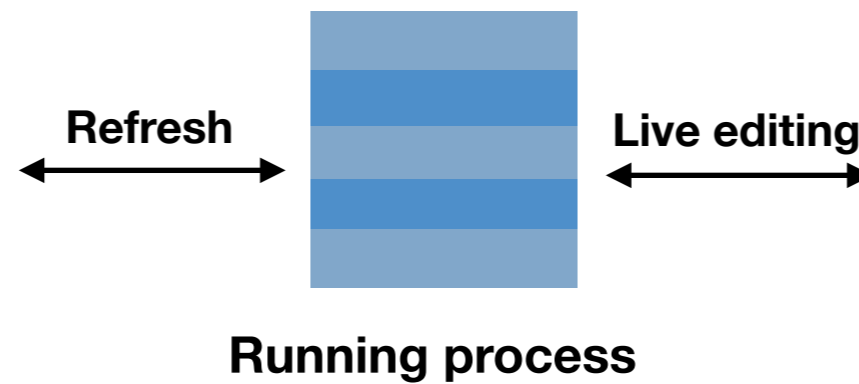


Install dependencies -> organize processes -> generate source -> compile -> package -> deploy

Live programming for better management UIs

Code.java:

```
object temperature extends Converter {  
  value1 = 0;  
  value2 := value1 * 9.0/5.0 + 32;  
  unit1 = "Celsius";  
  unit2 = "Fahrenheit";  
  title = "Temperature";  
}
```

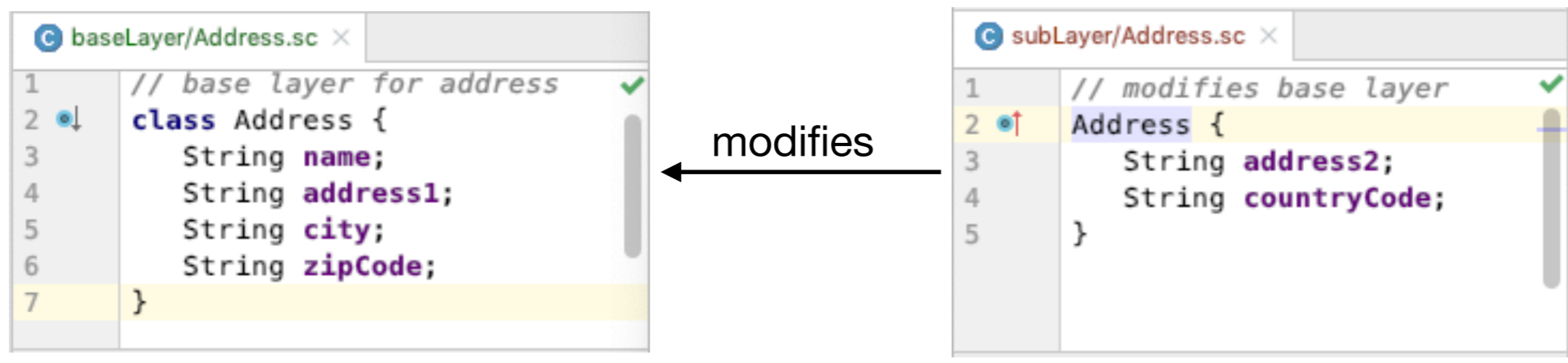


The screenshot shows a management UI for the 'temperature' object. It features a list of properties with corresponding input fields:

- object **temperature** extends **Converter**
- i** value1 =
- i** value2 :=
- S** unit1 =
- S** unit2 =
- S** title =

Layered languages - extensions to Java

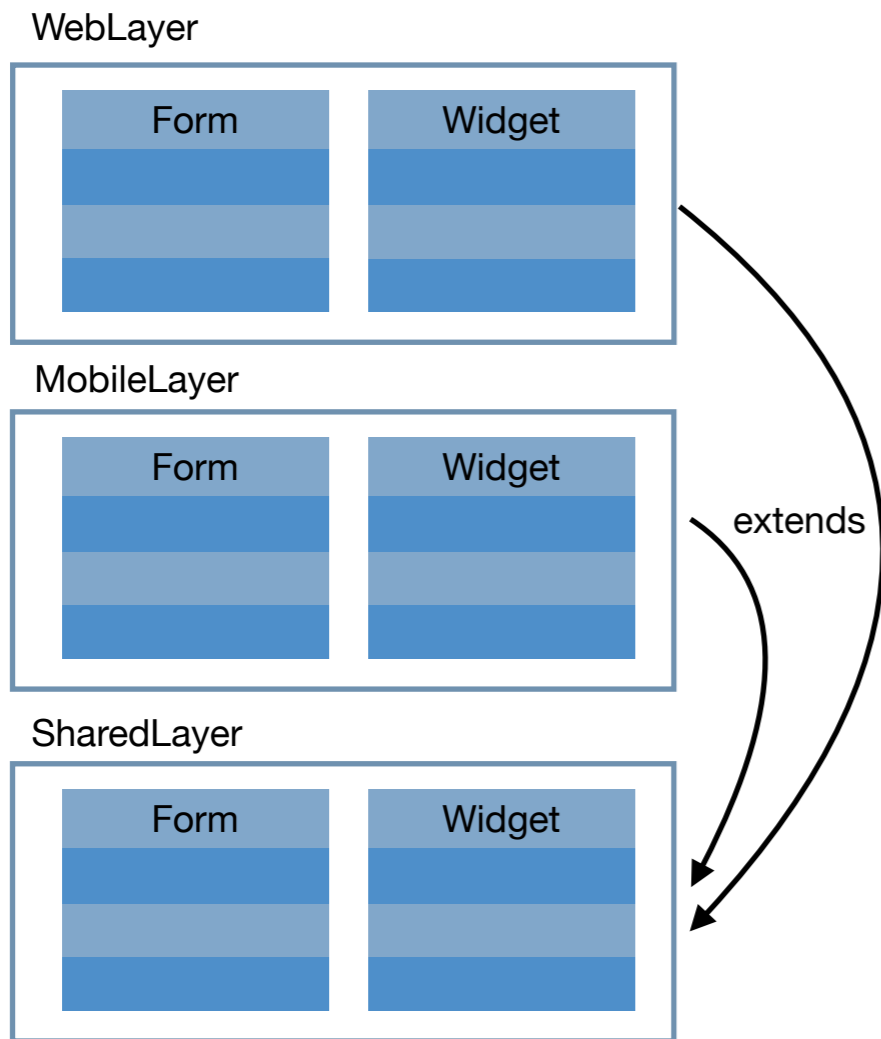
.sc -> StrataCode language



also: .sct, .shtml, .scxml, sccss, scsh formats - JSP-like integrated into the language for when a file is 'mostly text'

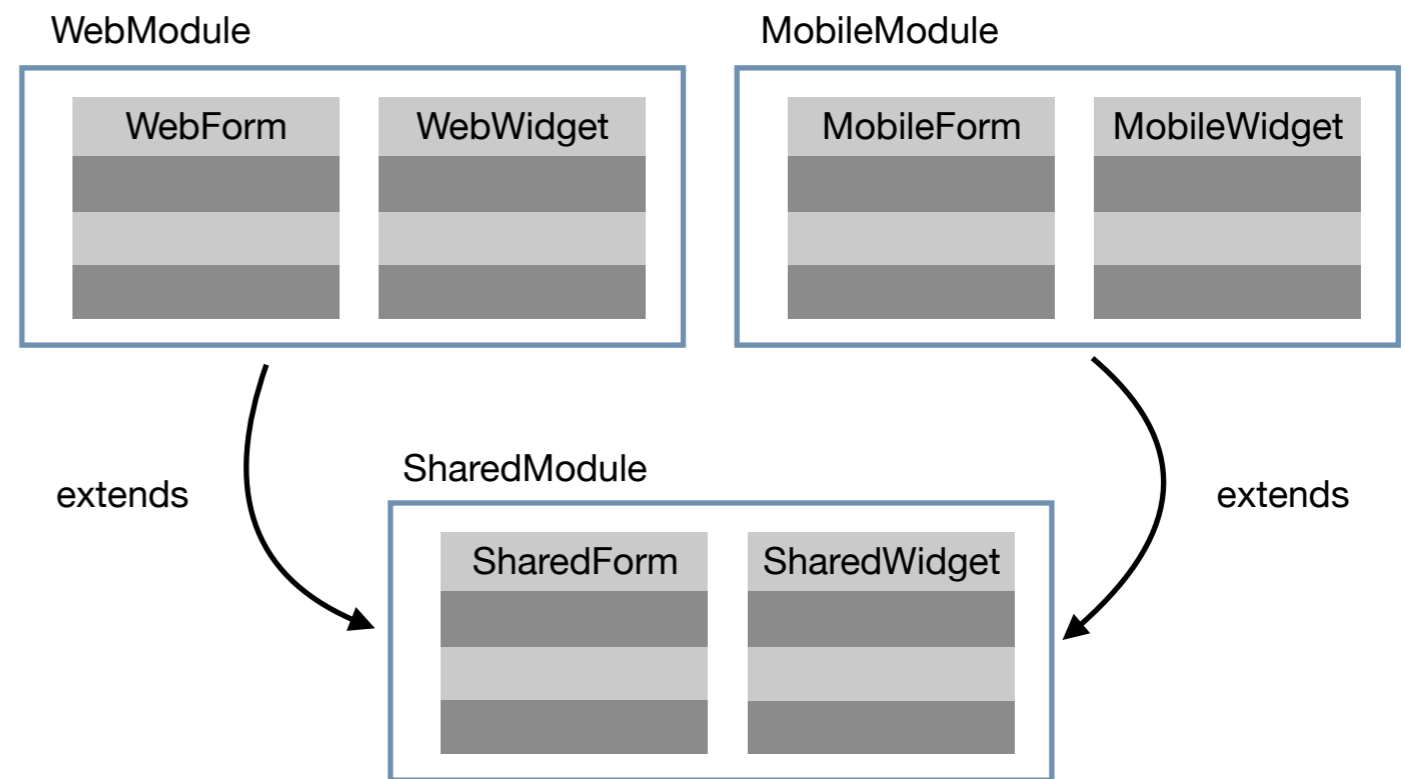
Separating code by dependencies

Layers using modify



- Code naturally organized by dependencies
- Better reuse, readability, refactorability

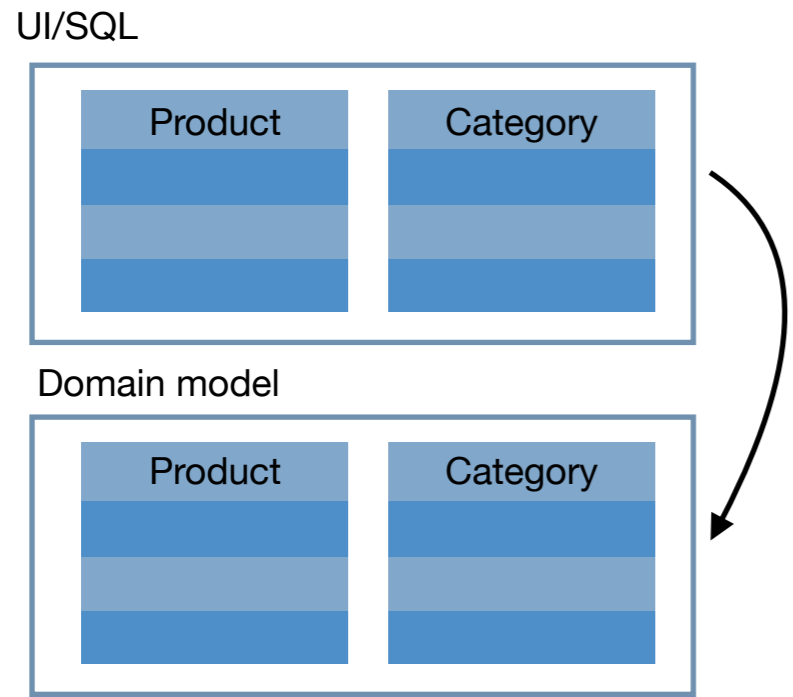
Modules using o/o inheritance



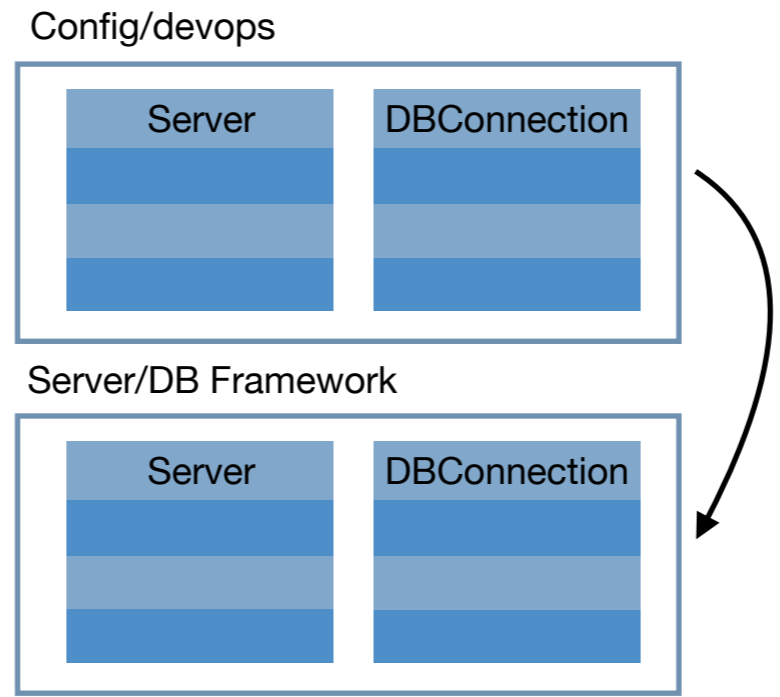
- More, longer type names
- Wrong type in code e.g. need WebForm but have SharedForm

Many uses for layers

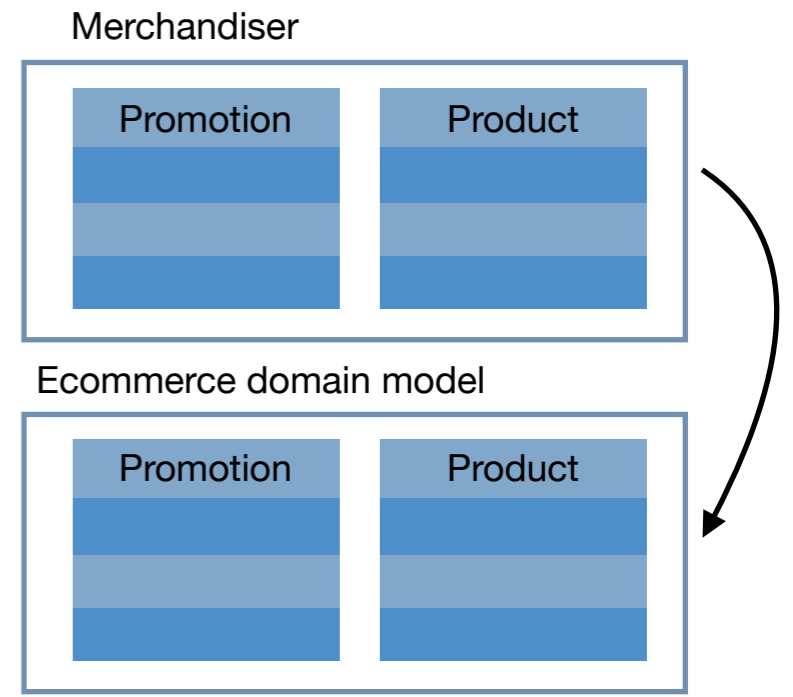
Separate UI/Persistence



Separate configuration



Business rules



And more: client/server, devops configuration + code, project configuration rules, testing, localization, style/design, plugins, inversion of control, 3rd party customizations, microservices, security sandboxes, dynamic/compiled code, A/B testing



Layered project organization

Layer definition file

```
modellImpl.sc x  html/.../core.sc x  schtml.sc x  lib.sc x  testScript.sc x  IdentifierExpression.java x  RemoteMethod.sc x  4
1 package sc.jetty;
2
3 jetty.lib extends servlet.lib, log4j.core {
4     compiledOnly = true;
5     hidden = true;
6     codeType = sc.layer.CodeType.Framework;
7
8     object jettyPkg extends MvnRepositoryPackage {
9         url = "mvn://org.eclipse.jetty/jetty-webapp/9.4.7.v20170914";
10    }
11
12    object jettySchemas extends MvnRepositoryPackage {
13        url = "mvn://org.eclipse.jetty.toolchain/jetty-schemas/3.1.RC0";
14    }
15
16    log4jPkg {
17        url = "mvn://org.slf4j/slf4j-log4j12/1.7.25";
18    }
19
20    public void init() {
21        // Exclude the javascript, android, and gwt runtimes, inherited by default for downstream layer
22        excludeRuntimes("js", "android", "gwt");
23
24        // Jetty for now is tied to the java_Server process.
25        // TODO: move this downstream so jetty.lib is usable in other processes
26        addProcess(sc.layer.ProcessDefinition.create("Server", "java", true));
27    }
28 }
29
```

- Written in dynamic StrataCode
- Static typed, IDE support
- Improve customization intent
- Simpler project directories

Features for declarative programming

Data binding

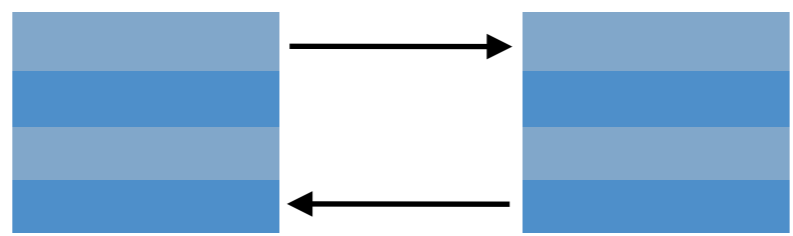
a := b
↑

a ::= b
↑ ↑

a =: b
└ ↑

a =: b()
└ ↑
eval

Components



init, start, validate

Properties

- get/set conversion
- change events
- mix compiled and dynamic properties in one type

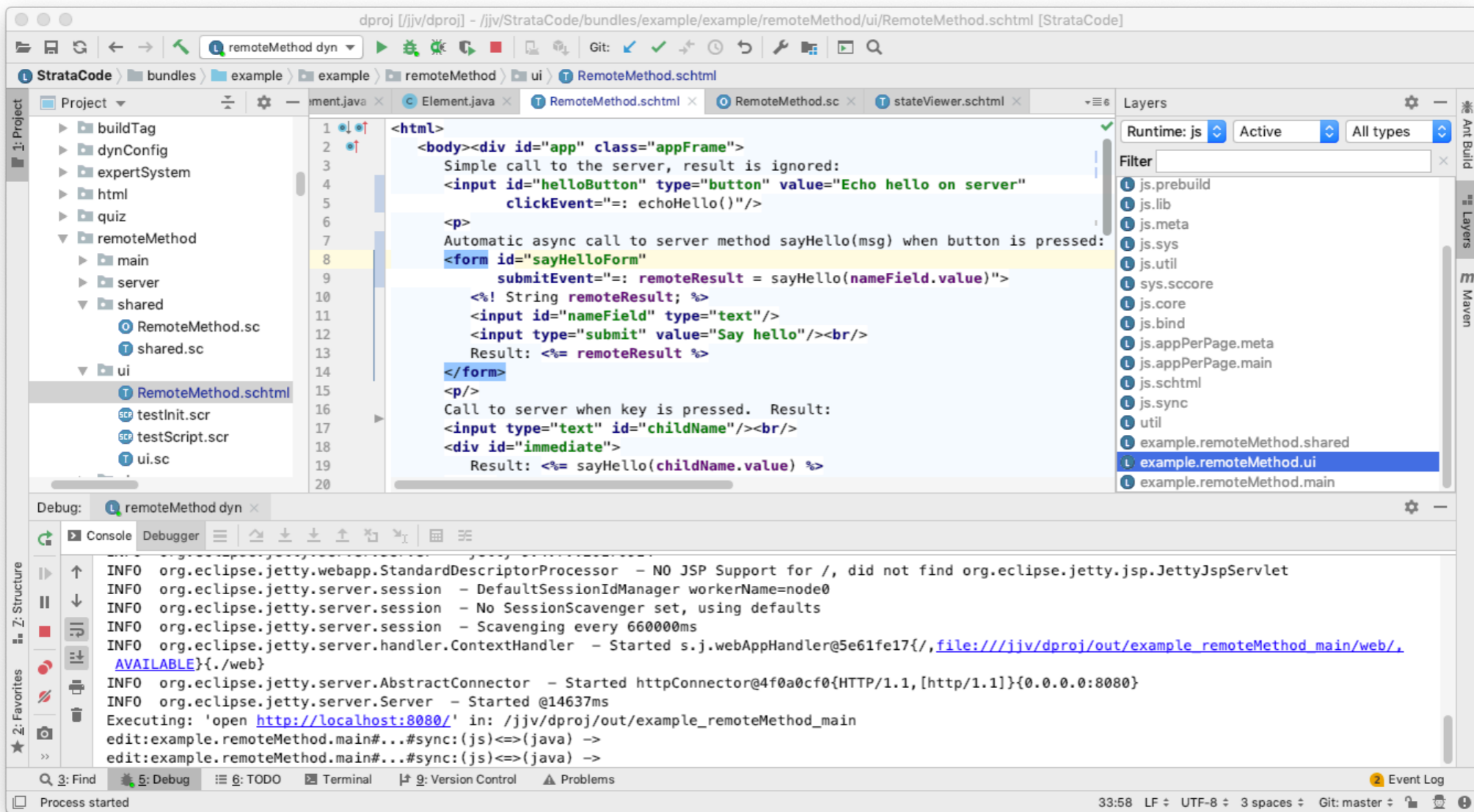
Templates

- dynamic text - sct, scxml, sccss,
- build languages on top (like schtml),
- stateful and stateless support
- JSP operators, but more like an extension to Java



IntelliJ plugin

Java-like editing, debugging for sc, sct, schtml, scj, scr in all frameworks





Code processing of language features

Annotations

Perform code processing on a type when an annotation is set - 'annotation layers' for compiled Java

parent/child relationships

Implement nested objects with a 3rd party library - code templates for compiled, IDynManager for dynamic

Full featured API, code processing engine, with runtime support

Supports compiled or source type systems. Full type indexing for both IDE or runtime. Optional 'liveDynamicTypes' mode to track object instances of certain types for management UIs.

IntelliJ plugin support built in

Usually no extra work to support framework features in the IDE

Much more - carefully designed hooks for framework developers



Current frameworks

3rd party integrations

android, swing, junit, jdbc, servlets, opengl, opencv, jetty, jpa,

Experimental: wicket, gwt

StrataCode web framework



StrataCode web framework

Rule oriented templates

```
<form submitEvent="=: addTodoEntry()">  
  <input type="text" value="=: todoText"/>  
  <input type="submit" value="Add" disabled=':= todoText.length() == 0' />  
</form>
```

Stateless

```
<ul>  
  <% for (int i = 0; i < 3; i++) { %>  
    <li><%= i %></li>  
  <% } %>  
</ul>
```

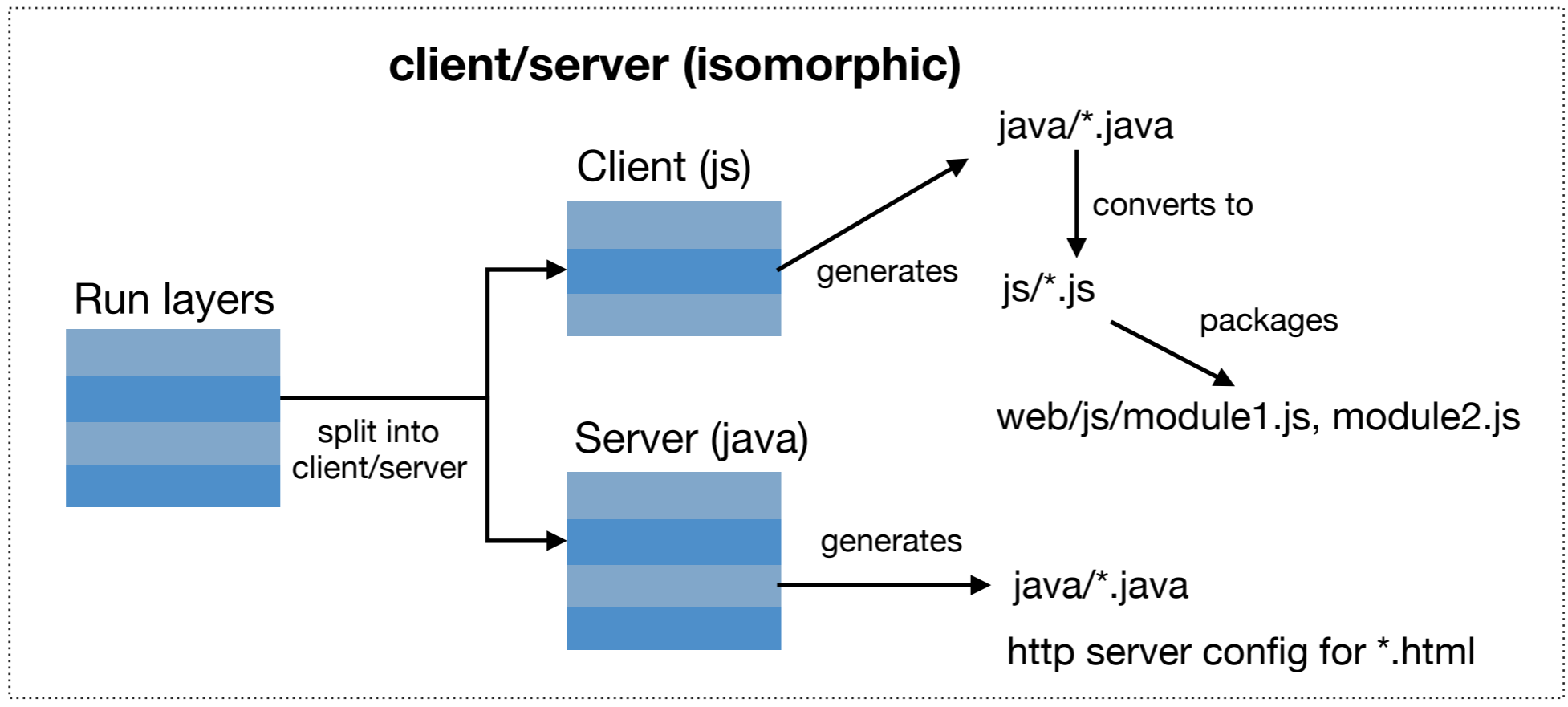
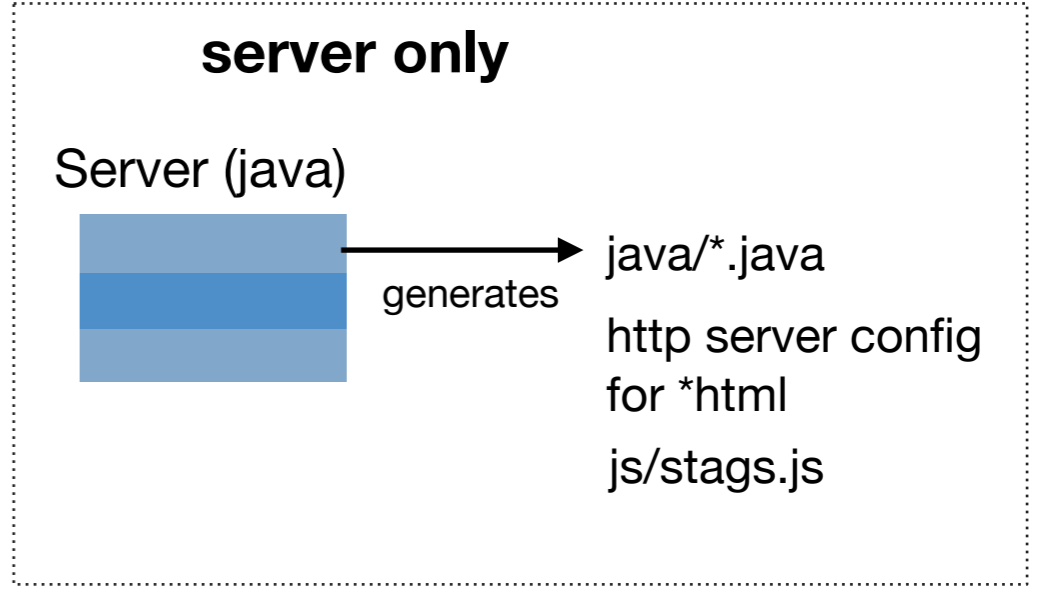
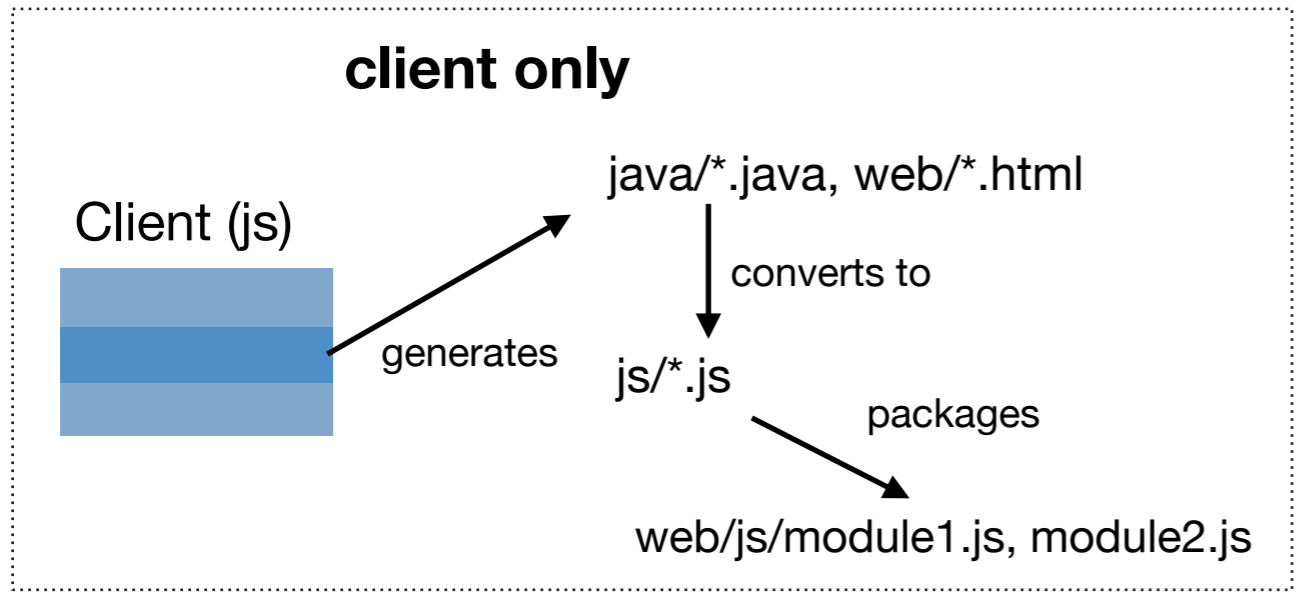
Converted to an output method

Stateful

```
<%=! ArrayList<Integer> vals = {0, 1, 2}; %>  
<ul>  
  <li repeat="vals"><%= repeatVar %></li>  
</ul>
```

Converted to reactive components

Three ways to deploy web components



Special tag attributes

extends - inherit attributes + body from another tag

abstract - define tag macros

visible - add/remove tag from page

class, style - set to expressions for dynamic logic

repeat - iterate tag

replaceWith - substitute a different tag

DOM events - click, mouseDown/Up/Move, keyDown/Up, focus/blur

DOM properties - clientWidth/Height, offsetTop/Left/Width/Height

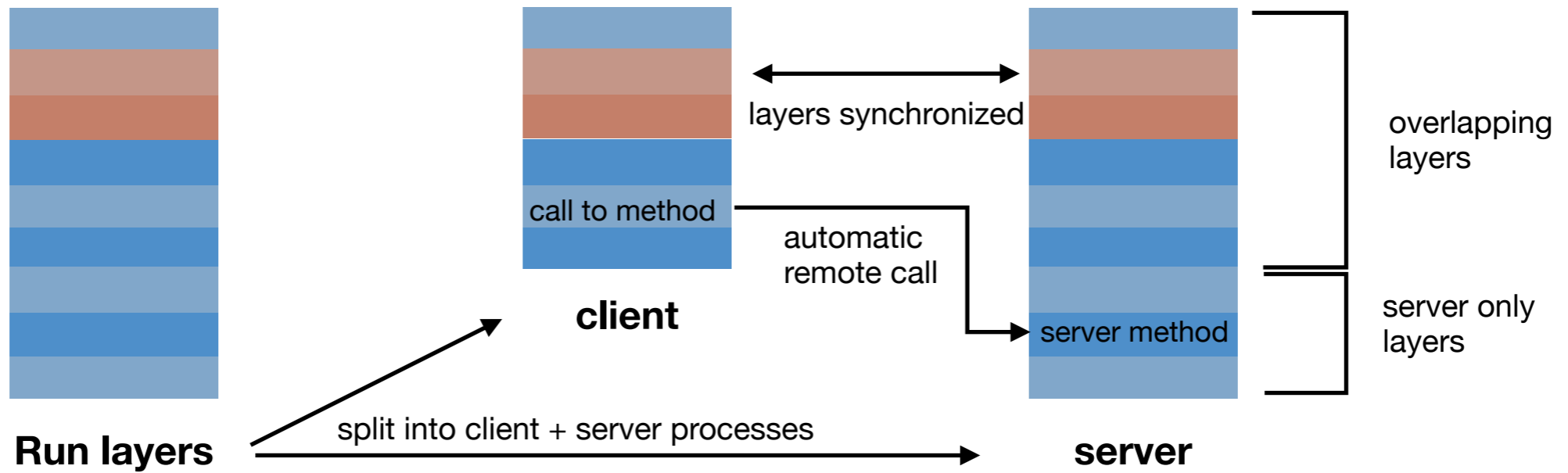
Merging - tagMerge, bodyMerge, addBefore, addAfter, orderValue

scope - change lifecycle of the tag/page: e.g. request, window, appSession, ...

exec - run tag on client, server, or both - by default inherits from parent tag or app default

(and much more)

Data synchronization + auto RPC



Async call with reverse data binding

```
Automatic async call to server method sayHello(msg) when button is pressed:
<form id="sayHelloForm" submitEvent="=: remoteResult = sayHello(nameField.value)">
  <%= String remoteResult; %>
  <input id="nameField" type="text"/>
  <input type="submit" value="Say hello"/><br/>
  Result: <%= remoteResult %>
</form>
```



Flexible runtime that evolves and scales efficiently

One syntax - two ways to run layers and types:



dynamic

on the fly changing code, runtime config, optimized for 'run once'

compiled

readable, debuggable Java

Mix compiled and dynamic features in one type - change the boundary as needed



Management UI framework

- Build UIs from domain objects
- Portable: desktop, web
- Edit configuration, rules - in place or in a new layer
- Create instances, types, properties, layers
- Navigate by type name, by layer or both
- Layers - multiple views on the same type

The screenshot displays the Strata Code Management UI framework interface. The interface is divided into several sections:

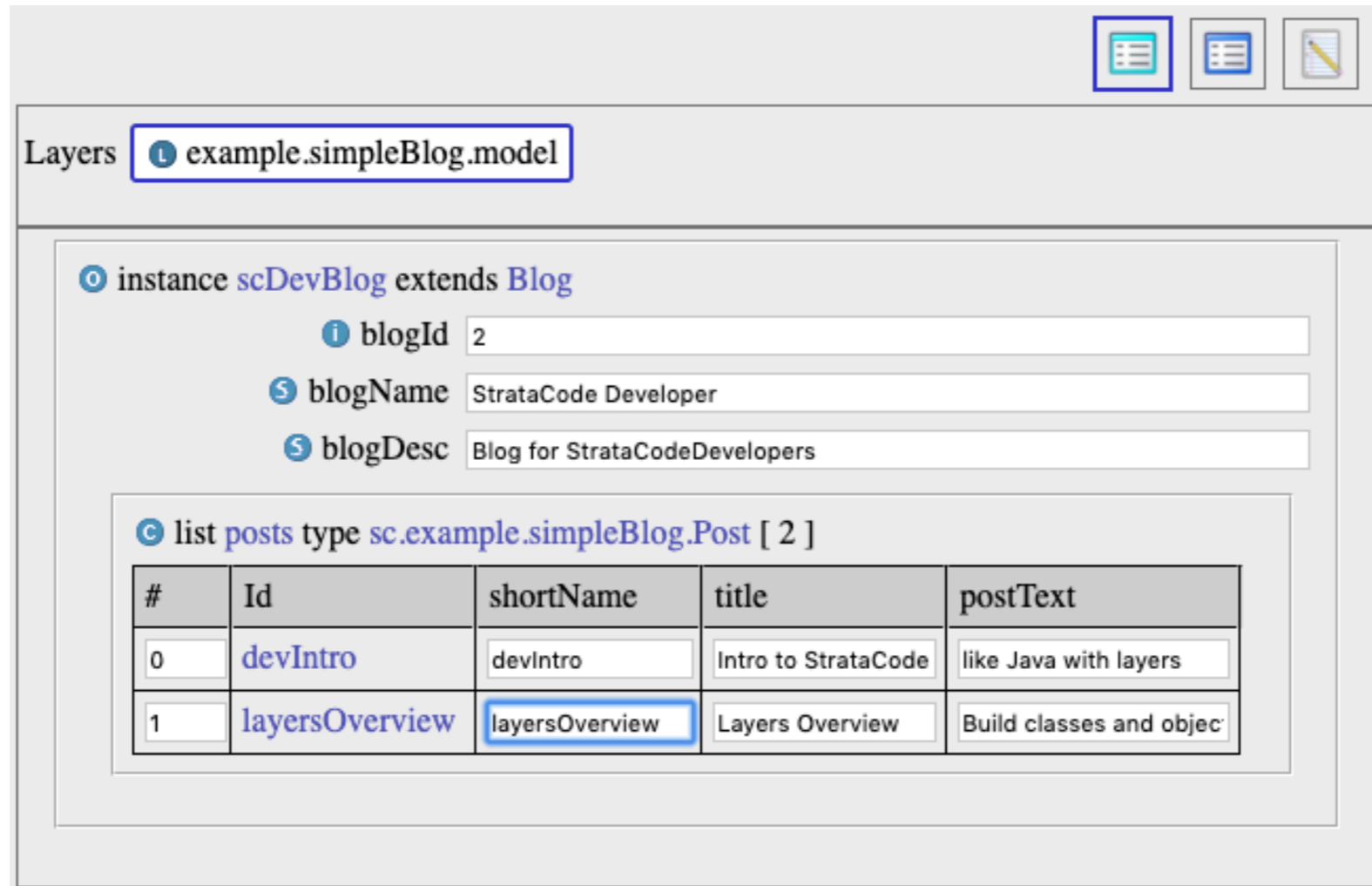
- All Types:** A tree view showing the hierarchy of types. The current selection is `sc.example.simpleBlog.model`.
- By Layer:** A tree view showing the hierarchy of layers. The current selection is `example.simpleBlog.model`.
- Layers:** A configuration panel for the selected layer. It shows an instance `scDevBlog` extending `Blog`. The configuration includes:
 - `blogId`: 2
 - `blogName`: StrataCode Developer
 - `blogDesc`: Blog for StrataCodeDevelopers
- Data Table:** A table showing a list of posts of type `sc.example.simpleBlog.Post`. The table has 2 rows and 5 columns: #, Id, shortName, title, and postText.

#	Id	shortName	title	postText
0	devIntro	devIntro	Intro to StrataCode	like Java with layers
1	layersOverview	layersOverview	Layers Overview	Build classes and objec

At the bottom of the interface, there is a search bar with the text `layersOverview` and a filter `String shortName = "layersOverview"`. There are also green and red checkmarks next to the filter.

Instance View

View, edit, create instances in the running application



Layers **example.simpleBlog.model**

instance scDevBlog extends Blog

- blogId** 2
- blogName** StrataCode Developer
- blogDesc** Blog for StrataCodeDevelopers

list posts type sc.example.simpleBlog.Post [2]

#	Id	shortName	title	postText
0	devIntro	devIntro	Intro to StrataCode	like Java with layers
1	layersOverview	layersOverview	Layers Overview	Build classes and objec



Type view

Edit property initialization, data binding rules, add properties
Updates source files incrementally for mixed tool/developer workflows

The screenshot shows the Strata Code IDE interface. On the left, there is a 'Temperature' widget with input fields for Celsius (0) and Fahrenheit (32). The main workspace is divided into several panels:

- All Types:** A tree view showing the project structure. Under 'example', 'unitConverter' is expanded to show 'UnitConverter' (selected), 'Converter', 'body', 'converters', 'head', and 'numberConverter'.
- By Layer:** A tree view showing the layer structure. Under 'example', 'unitConverter' is expanded to show 'coreui', 'UnitConverter' (selected), 'html', 'clientServer', 'core', 'model', and 'UnitConverter'.
- Layers:** A list of layers: 'example.unitConverter.html.core', 'example.unitConverter.coreui', and 'example.unitConverter.model'.
- Class Definition:** The 'class UnitConverter' is shown with two objects:
 - object converters** extends **ComponentList**:
 - empty =
 - object temperature** extends **Converter**:
 - value1 =
 - value2 :=:
 - unit1 =
 - unit2 =
 - title =
 - object distance** extends **Converter**:
 - value1 =
 - value2 :=:
- Bottom Panel:** A table-like view showing the binding for the 'temperature' property: 'double value2' with the binding rule 'value1 * 9.0 / 5.0 + 32'. It includes a dropdown menu and validation icons (green checkmark and red X).



Code view

Mini IDE (using codemirror in the browser, rtext in swing)
Edit-time errors, syntax highlighting, code-hinting

The screenshot shows a web-based code editor interface. At the top right, there are three icons: a list, a document, and a pencil. Below these is a "Layers" section with a dropdown menu showing "example.simpleBlog.model". The main editor area has a tab labeled "Source" and a file path "blogs/scNewsBlog.sc layer: example.simpleBlog.model(sc.example.simpleBlog)" with a green checkmark and a red X icon. The code is as follows:

```
1 import sc.example.simpleBlog.blogs.news.*;
2
3 object scNewsBlog extends Blog {
4   blogId = 1;
5   snogName = "StrataCode Announcements";
6   blogDesc = "Announcements from StrataCode";
7
8   posts = { toBeAnnounced, noNewsGoodNews };
9 }
10
```

Navigate by type or by layer

All Types

- sc
 - example
 - unitConverter
 - + ○ UnitConverter

By Layer

- example
 - unitConverter
 - ● coreui
 - + ○ UnitConverter
 - html
 - + ● clientServer
 - ● core
 - + ○ UnitConverter
 - ● model
 - + ○ UnitConverter

← merged view

← file system view



Versatile test scripts, command line

```
Product.sc x TypeEditor.java x testEditBooks.scr x testToDoList.scr x model/ToDoList.sc x 5
1 cmd.pauseTime = 250;
2
3 // Run these commands in the java process - they will be sync'd to the client.
4 cmd.targetRuntime = "java";
5 // Optionally - send the commands to the client - after converting to JS,
6 // they are eval'd using the sync framework.
7 //cmd.targetRuntime = "js";
8
9 ToDoList {
10 todos.size();
11 assert todos.size() == 3 : "No todos";
12 todos.get(1).complete = true;
13
14 todoText = "A new entry";
15 addTodoEntry();
16 removeComplete();
17
18 while (todos.size() > 0) {
19     todos.get(0).complete = true;
20     removeComplete();
21 }
22
23 todoText = "done";
24 addTodoEntry();
25 cmd.sleep(1000);
26 todos.get(0).complete = true;
27 }
28
```

- Line-oriented StrataCode
- IDE support
- Target one or more processes
- Automatic remote methods
- Layering, nesting with 'include'
- Script mode - edit instances
- Edit mode - edit types



Learn more

Learn more at www.stratacode.com

Contact jeff@jvroom.com

See the status page for how we are doing

Examples, documentation, articles

Ideas for improvements?

Build something together?