Fluid Class Definitions

Marcus Denker, Inria Evref

Talk a bit late

- Was for 2024 (and too late already)
- But too many talks @ ESUG
- But still I think important to have

!#? What

- Why did you change that ?

```
Object << #Point
    slots: { #x . #y };
    tag: 'BasicObjects';
    package: 'Kernel'
```

Class Definition

- Classes are defined with an expression
- Code is *not* read from disk
    - But instead pretty printed from the class data itself
- But we do log in .sources/changes

ST80 style

```
Object subclass: #Point
	instanceVariableNames: 'x y'
	classVariableNames: ''
	package: 'Kernel-BasicObjects'
```

Message send

not optional

String

Properties

- Message to the superclass
  - lots of arguments
- When evaluated, class is created and installed as side effect
- Some things would be nice:
  - Variables not strings
  - Simpler (optional parameters)

Problem: lots of methods

- Using methods with lots of args does not scale
- Already ST80 had *many* methods
- We added many more

Examples Pharo11

- Traits
- Optional pools
- Package vs category
- slots vs instanceVariable

Layouts

- In ST80: layout in the selector
- we have to copy *many* methods for each layout
  - weak
  - byte, doubleByte
  - Immediate
  - variable, variableByte, variableWord

Layouts: even more

- Ephemerons
- CompiledMethod
- There is no way to create a class like that!

```
variableByteSubclass: t instanceVariableNames: f classVariableNames:
poolDictionaries: s package: cat

    | oldClassOrNil actualLayoutClass |
    oldClassOrNil := self environment at: t ifAbsent: [ nil ].
    actualLayoutClass := (oldClassOrNil notNil and: [ oldClassOrNil
classLayout class == CompiledMethodLayout ])
        ifTrue: [ CompiledMethodLayout ]
        ifFalse: [ ByteLayout ].
```

This does not scale!

- We have a combinatorial explosion
- In Pharo 11 we had > 30 methods already
    - And many combinations are missing

Solution: Fluid

- Fluid class Definitions. Default since Pharo 12

```
ByteArray << #CompiledCode
    layout: CompiledMethodLayout;
    slots: {};
    sharedVariables: { #LargeFrame . #SmallFrame .
            #PrimaryBytecodeSetEncoderClass .#SecondaryBytecodeSet
            #EncoderClass };
    tag: 'Methods';
    package: 'Kernel-CodeModel'
```

Cascade

optional

Solution: Fluid

- Uses classBuilder

```
ByteArray << #CompiledCode
    layout: CompiledMethodLayout;
    slots: {};
    sharedVariables: { #LargeFrame . #SmallFrame .
            #PrimaryBytecodeSetEncoderClass .#SecondaryBytecodeSet
            #EncoderClass };
    tag: 'Methods';
    package: 'Kernel-CodeModel'
```

Returns a classBuilder

- you need to send #install

What it gets you

- Easy to make parameters optional (e.g. class variables)
- easy to add new parameters
- We used this already a lot

Easy to evolve

- Fluid class Defintions. Default since Pharo12

```
ByteArray << #CompiledCode
    layout: CompiledMethodLayout;
    slots: {};
    sharedVariables: { #LargeFrame . #SmallFrame .
            #PrimaryBytecodeSetEncoderClass .#SecondaryBytecodeSet
            #EncoderClass };
    tag: 'Methods';
    package: 'Kernel-CodeModel'
```

Explicit Layout

not strings

ClassLayout

- The layout is now a parameter
- Supports new Ephemeron layout
- CompiledMethodLayout problem solved
- We can now start to experiment with defining new layouts
  - Example: BitPattern layout

SharedVariables

- SharedVariable / ClassVariable definition is not a String
- You can define your own !

```
Object << #MyClass
    slots: {};
    sharedVariables: { #Variable => WeakClassVariable };
    package: ‘Example'
```

The Class Builder

```
ByteArray << #CompiledCode
    layout: CompiledMethodLayout;
    slots: {};
    sharedVariables: { #LargeFrame . #SmallFrame .
            #PrimaryBytecodeSetEncoderClass .#SecondaryBytecodeSet
            #EncoderClass };
    tag: 'Methods';
    package: 'Kernel-CodeModel'
```

This returns a classBuilder

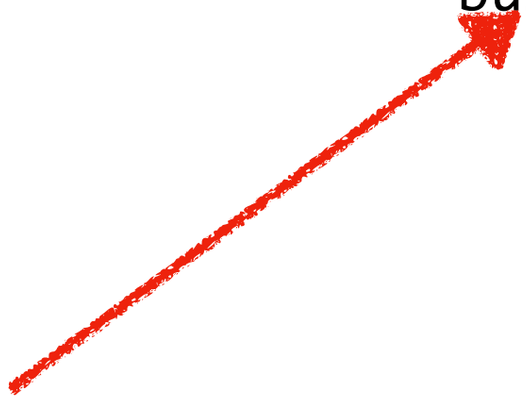
Class Builder

- Can be used programmatically, too
    - Makes it much easier to use
    - Copy paste!
- Very nice in combination with classInstaller

Example

- From testAddingATraitToAClassHasRightProtocolsOnMetaclass

```
newClass := ShiftClassInstaller make: [ :builder |
                    builder
                        name: #SHClass;
                        traits: { TViewModelMock };
                        package: self generatedClassesPackageName ].
```



Here we get the builder, too

Backward compatibility

- CodeImporter can still read old definitions
  - e.g. load old .cs /.st files
- uses OldClassDefinitionBuilder

Fluid: Enables even more

- Fluid is an investment in the future
- It enables to experiment much easier
- Some examples

Example: #isAbstract

- Now we implement #isAbstract on the class side
- But it is declarative
  - using code to model declarative information is bad!
- Fluid: We can extend the Class Definition instead!

Class Deprecation

- We support Class Deprecation, again not declarative
  - #isDeprecated
  - #deprecatedAliases:

Questions?

This returns a classBuilder

```
ByteArray << #CompiledCode
    layout: CompiledMethodLayout;
    slots: {};
    sharedVariables: { #LargeFrame . #SmallFrame .
            #PrimaryBytecodeSetEncoderClass .#SecondaryBytecodeSet
            #EncoderClass };
    tag: 'Methods';
    package: 'Kernel-CodeModel'
```

Cascade

optional