

Mixins

Groovy Mixin Proposal

Take the following classes.

Wombat.groovy

```
class Wombat {

    int getStrength() { 5 } // small, weak
    int getIntelligence() { 2 } // not too
swift this way either
    int getDexterity() { 6 } // not to swift
    int getConstitution() { 14 } // wombats
are healthy
    int getWisdom() { 2 } // poor decision
making
    int getCharisma() { 15 } // wombats are
cute
    int getHitDice() { 2 }

    /**
     * wombats turn very quickly for their
size and agility
     */
    getTurningSpeed() {
        5
    }
}
```

Mobile.groovy

```
class Mobile {
    heading = 0
    locationX = 0
    locationY = 0

    walk(seconds) {
        distance = getWalkingSpeed() *
seconds
        locationX = distance *
applyTrigIDontRememberTo(heading)
        locationY = distance *
applyMoreTrigIDontRememberTo(heading)
        distance
    }

    turnRight(seconds) {
        heading += seconds *
getTurningSpeed()
    }

    turnLeft(seconds) {
        heading -= seconds *
getTurningSpeed()
    }

    double getTurningSpeed() {
        ((getDexterity() +
getIntelligence()) / 2) / 3
    }
}
```

```
    getWalkingSpeed() {  
        (((getStrength() + getDexterity()) /  
2) / 3) * 10  
    }
```

```
abstract getDexterity();
```

```
abstract getStrength();
abstract getIntelligence();
}
```

FighterType.groovy

```
class FighterType implements CharacterClass
{
    getThaco() {
        20 - getHitDice() -
        (Bonuses.getStrengthModifier(getStrength()))
    }

    abstract getHitDice();
    abstract getStrength();
}

interface CharacterClass {
    int getThaco();
}
```

The Wombat class as defined simply defines a few things. By defining those things it makes it possible for it to mixin other things - notable Mobile and FighterType.

Now, how to mix these things together...

Static Mixing

At compile time mixin information can be provided, in which case all instances of the class receiving the mixin have the mixed traits

Wombat.groovy

```
class Wombat {  
    mixin(Mobile)  
    ...  
}
```

This says that all instances of Wombat are Mobile. An alternate for this is to put it on the class declaration line:

Wombat.groovy

```
class Wombat with Mobile {  
    ...  
}
```

Which borrows the keyword/syntax from Scala. I lean towards the first because Java allows you to cast to things in the type declaration, and you specifically cannot cast to Mobile in this case. The `mixin(Wombat)` would be treated as an initializer block

All methods defined in Mobile would be added to Wombat. The type information is not. If a method is already defined in Wombat (such as `getTurningSpeed()`) the mixin method is **not** mixed in. Similarly, if multiple mixins are specified, the first time a method is defined is the one that is used.

Instance variables do not cross declarations. That means that Wombat cannot directly access heading, it must go through `getHeadin()`. Similarly, the mixed in class cannot access instance variables on the class it is being mixed into.

Mixins can define their own internal instance variables, only accessible to the mixed in class itself (heading, locationX, locationY).

"this" refers to the same instance whether used in the mixed in class or the mixed class.

Methods the mixed in instance requires from the host instance can be declared abstract. It is worth considering if they can also be left undeclared and then invoked via dynamic dispatch - this is open for discussion. It is useful to document the methods the mixin requires, though, so I used the abstract declaration here.

Dynamic Mixing

Sample.groovy

```
george = new Wombat()  
george.mixin(FighterType)
```

or

Sample.groovy

```
george = new Wombat().mixin(FighterType)
```

adds the `FighterType` mixin to the `Wombat` instance referenced by `George`. `FighterType` is only available on that particular instance, so:

Wombat.groovy

```
george = new Wombat().mixin(FighterType)  
william = new Wombat()  
  
george.getThaco() // returns 19 (20 - 2 -  
(-1))  
william.getThaco() // error
```

Singleton Method Mixin

Adding a single method at runtime has a bit of syntactic sugar, as follows:

Sample.groovy

```
george = new Wombat()
george.mixin("say") { |something| println
something }
george.say("Hello, world!")

// notice closure executes in context of
george, with the mixed in say(..) method
george.mixin("sayHello") { say("hello") }
george.sayHello()
```

Paramaterized Mixing

The mixin class is allowed to use a constructor which takes args, so:

MyMixin.groovy

```
class MyMixin {
    MyMixin(thing) { ... }
}

anArg = "wibble!"
george.mixin(MyThing, anArg)
```

which would initialize the mixin state via the ctor giving it anArg as a lone ctor arg