Eight Ceylon Idioms

Gavin King - Red Hat

profiles.google.com/gavin.king

ceylon-lang.org

Ceylon is:

Find more bugs at compile time, thanks to scrupulous use of static typing

Powerful,

More powerful tools for abstraction, so the static types get in the way less

We spend much more time reading other people's code than we spend writing our own

Readable,

Verbosity is not the only relevant measure, but it does matter

The developer should be able to reproduce the reasoning of the compiler according to intuitive rules

Predictable.

Error messages should be understandable and refer to concepts inside the user's head

Ceylon has:

Cross platform execution:
JVM or JavaScript VM

A Platform,

A brand new modular SDK with a minimal language module

Static typing enable powerful tools – dynamic languages simply can't compete

Tooling,

The job of tools isn't to fill in boilerplate, it is to help you understand and evolve the code

A single unifying module system built into the language, the tooling, the compiler, the runtime

Modularity.

Ceylon Herd: the community module repository

Idiom #1

Idiom: functions with multiple outcomes

For example, an operation might return a `Person`, an `Org`, or nothing.

```
//Java
Object findByName(String name)
        throws NotFoundException { ... }
```

We can handle the different outcomes using `instanceof`, type casts, and `catch`:

```
try {
    Object result = findByName(name);
    if (result instanceof Org) {
        Org org = (Org) result;
        ...
    }
    if (result instanceof Person) {
        Person person = (Person) result;
        ...
    }
}
catch (NotFoundException nfe) { ... }
```

Idiom #1

Idiom: functions with multiple outcomes

A function with more than one “outcome” can be defined using a union type.

```
Org|Person|NotFound findByName(String name) => ... ;
```

We can handle the various outcomes using `switch`:

```
value result = findByName(name);
switch (result)
case (is Org|Person) { ... }
case (is NotFound) { ... }
```

Item? get(Key key)

Idiom #2

Idiom: functions returning null

Example: retrieving an element from a map. (Special case of multiple outcomes!)

```
Item? get(Key key) => ... ;
```

For a union type of form `Null|Item`, we have some special syntax sugar:

```
value map = HashMap { “enrique”->cst, “tom”->gmt, “ross”->pst };
```

```
Timezone tz = map[name] else cet;
```

item? get(Key key)

Idiom #2

Idiom: functions returning null

What if we *know* that `get()` can't return `null`, because of some constraint upon the given key? We can make use of an assertion.

```
if (name in map) {
    assert (exists tz = map[name]);
    return ZonedTime(time, tz);
}
```

A different way to write this code:

```
if (exists tz = map[name]) {
    return ZonedTime(time, tz);
}
```

Idiom #3

Idiom: overloading

For example, an operation might apply to an `Integer`, or to a `Float`.

```
//Java
Float sqrt(Float number) { ... }
Float sqrt(Integer number) { ... }
```

Actually two different, unrelated operations.

Idiom #3

Idiom: overloading

A function parameter can be defined using a union type.

```
Float sqrt(Float|Integer number) {
    switch (number)
    case (is Float) { ... }
    case (is Integer) { ... }
}
```

Now we have a single operation:

```
Float fourthRoot(Float|Integer number) => sqrt(sqrt(number));
```

And we can obtain a reference to it:

```
Float(Float|Integer) sqrtFun = sqrt;
```

Idiom #4

Idiom: multiple return values

For example, an operation might return a `Name` and an `Address`.

```
//Java
class NameAndAddress { ... }

NameAndAddress getNameAndAddress(Person person) {
    return new NameAndAddress(new Name(person.getFirst(), person.getLast()),
                              person.getHome().getAddress());
}
```

We *have to* define a class.

Idiom #4

Idiom: multiple return values

A function can be defined to return a tuple type.

```
[Name,Address] getNameAndAddress(Person person)
        => [Name(person.first, person.last),
            person.home.address];
```

Now a caller can extract the individual return values:

```
value nameAndAddress = getNameAndAddress(person);
Name name = nameAndAddress[0];
Address address = nameAndAddress[1];
```

What about other indexes?

```
Null missing = nameAndAddress[3];
Name|Address|Null val = nameAndAddress[index];
```

Idiom #5

Idiom: spreading tuple return values

Imagine we want to pass the result of `getNameAndAddress()` to another function or class initializer.

```
class SnailMail(Name name, Address address) { ... }
```

We can use the spread operator, *, like in Groovy:

```
value snail = SnailMail(*getNameAndAddress(person));
```

Or we can work at the function level, using `unflatten()`

```
SnailMail(Person) newSnail
        = compose(unflatten(SnailMail), getNameAndAddress);
SnailMail snail = newSnail(person);
```

Idiom #6

Idiom: unions of values

Imagine we want to write down the signature of `Set.union()` in Java:

```
//Java
interface Set<T> {
    public <U super T> Set<U> union(Set<? extends U> set);
}
```

This doesn't actually compile since Java doesn't have lower bounded type parameters. (The equivalent thing would work in Scala though.)

```
Set<Foo> setOfFoo = ... ;
Set<Bar> setOfBar = ... ;

Set<Object> setOfFoosAndBars = setOfFoo.union(setOfBar);
```

Idiom #6

Idiom: unions of values

Unions of values correspond to unions of types!

```
interface Set<These> {
    shared formal Set<These|Those> union<Those>(Set<Those> set);
}
```

Exactly the right type pops out automatically.

```
Set<Foo> setOfFoo = ... ;
Set<Bar> setOfBar = ... ;

Set<Foo|Bar> setOfFoosAndBars = setOfFoo | setOfBar;
```

Idiom #7

Idiom: intersections of values

Now let's consider the case of `Set.intersection()` in Java.

```
//exercise for the audience
```

I tried a bunch of things and didn't come close to anything like the right thing.

Idiom #7

Idiom: intersections of values

Intersections of values correspond to intersections of types!

```
interface Set<These> {
    shared formal Set<These&Those> intersection<Those>(Set<Those> set);
}
```

Again, exactly the right type pops out automatically.

```
Set<Foo> setOfFoo = ... ;
Set<Bar> setOfBar = ... ;

Set<Foo&Bar> setOfFooBars = setOfFoo & setOfBar;
```

Idiom #7

Idiom: intersections of values

Example: the `coalesce()` function eliminates `null` elements from an `Iterable` object.

```
{Element&Object*} coalesce<Element>({Element*} elements)
        => { for (e in elements) if (exists e) e };
```

Exactly the right type pops out automatically.

```
{String?*} words = { “hello”, null, “world” };
{String*} strings = coalesce(words);
```

Idiom #8

Idiom: discovery

Example: auto-discover classes annotated `test`.

```
shared test class IdiomTests() { ... }
```

We use the Ceylon metamodel:

```
void metamodel() {
    value declarations =
        `package org.jboss.example.tests`
            .annotatedMembers<ClassDeclaration,TestAnnotation>();
    for (decl in declarations) {
        if (decl.parameterDeclarations.empty) {
            value model = decl.classApply<Anything,[]>();
            // instantiate the class
            print(model());
        }
    }
}
```

Where to go for more

Ceylon community

http://ceylon-lang.org

Ceylon on G+

http://ceylon-lang.org/+