Notes - 5/30

Wednesday, May 30, 2012 9:56 AM

[In 41/1727 at 10:00 today.]

Agenda:

- ES6 classes alignment discussion (attached)
- Details of short function syntax (attached)
- Recursive type inference rules
- Continue review of design items spreadsheet
- Others?

- Release (package, OSS, docs, samples)
- Scalability
- Quality & completeness
- Classes work items
  - Partner migration
  - Class expressions no
- Win8 (project and debug)

Classes:

-

```
class Foo {
 bar() {

 }
}
```

```
class Foo() {
  public y = bar();
  public bar() {
    var x = 4;
    this.x
    return this.y + x;
  }
}
```

75

It is seeming increasingly likely that the “maximally minimal classes” proposal will be adopted as ES6’s class syntax.

http://wiki.ecmascript.org/doku.php?id=strawman:maximally_minimal_classes

If max/min classes end up in ECMAScript, they’ll eventually end up in Strada too. So, rather than have two subtly different kinds of classes, it would be if Strada could adopt max/min classes as a baseline. Steve, Luke, and I have been contemplating what that might look like. Below is a strawman propos that we can discuss in the design meeting tomorrow.

Strada will extend the max/min class proposal with the following capabilities:

- Inference of class members from ‘this.xxx’ assignments.
- Type annotations.
- Properties initialized from ‘this.xxx’ constructor parameters.
- Explicit property declarations.
- public and private modifiers (the default being public).
- Static members.

Here’s an example of an ES6 class:

this

```
class Point
{
    x = this;
    constructor(x, y) {
        this.x = x;
        this.y = y;
        this.getDist = () => { this.z = 10; } // this is Point
        this.getDist = function() { this.z = 10; } // this is any
    }
    getDist = function() { return this.window; } // this is any
    getDist() { return Math.sqrt(this.x * this.x + this.y * this.y); }
    setTag(tag) { this.tag = tag; }
}
Point.prototype.x = 0;

function Point(x,y) {
  this.z = this.calc();
  this.x = x;
  this.y = y;
  this.getDist = () => { this.z = 10; } // this is Point
  this.getDist = function() { this.z = 10; } // this is any
}
Point.origin = new Point(0,0);
Point.prototype.calc = function() { return 3; }
Point.prototype.getDist = function() { this.z = 10; return this.x; }
```

```
declare var window : {
  onmousemove: (this: Element, ev: MouseEvent) => void;
}
window.onmousemove = function(ev) {
  this.
}
```

```
function Point() {
  this.x
}
Point.prototype = {
  getDist: function() {
```

```
        this.x;
      }
    }
```

Two really big differences from Strada's current class implementation is that instance members are *not* explicitly declared and members are *not* automatically in scope (members can only be accessed through 'this'). Since there are no explicit instance member declarations, Strada will have to inf instance members from the 'this.xxx' assignments in the class body.

From the 'this.xxx' assignments in the declaration above, Strada infers three public properties, x, y, and tag, all of type any. Strada makes inferences fr 'this.xxx' assignments in the class body, not just assignments in the constructor. However, local function declarations that introduce a new meaning fo are ignored.

When there are multiple 'this.xxx' assignments to the same property, Strada infers the 'best common type' (exactly the same algorithm we use to infe return types from multiple return statements).

Here is the example with type annotations. With these annotations, the x, y, and tag public properties are inferred as number, number, and string respectively.

```
class Point
{
    constructor(x: number, y: number) {
        this.x = x;
        this.y = y;
    }
    getDist() { return Math.sqrt(this.x * this.x + this.y * this.y); }
    setTag(tag: string) { this.tag = tag; }
}
```

Properties can be automatically initialized from constructor parameters using the 'this' keyword:

```
class Point
{
    constructor(this.x: number, this.y: number) { }
    getDist() { return Math.sqrt(this.x * this.x + this.y * this.y); }
    setTag(tag: string) { this.tag = tag; }
}
```

Automatic initialization from a constructor parameter simply corresponds to a 'this.xxx = xxx' assignment in the constructor and is processed as such fo purposes of type inference.

Properties can be explicitly declared in the class body. Initializers, if present, are executed once at class construction time and the properties are create the prototype object (similar to methods). For this reason, initializer expressions cannot refer to constructor parameters.

```
class Point
{
    x = 0;
    y = 0;
    constructor(x: number, y: number) {
        if (x != 0) this.x = x;
        if (y != 0) this.y = y;
    }
    getDist() { return Math.sqrt(this.x * this.x + this.y * this.y); }
    setTag(tag: string) { this.tag = tag; }
}
```

If a property is explicitly declared, the type specified or inferred in the explicit declaration becomes the type of the property and no inferences are mad from 'this.xxx' assignments.

Member declarations may include a 'public' or 'private' modifier. The default is 'public'. Here is the example using private properties:

```
class Point
{
    private _x = 0;
    private _y = 0;
    constructor(x: number, y: number) {
```

```
        this._x = x;
        this._y = y;
    }
    get x() { return this._x; }
    get y() { return this._y; }
    getDist() { return Math.sqrt(this.x * this.x + this.y * this.y); }
    setTag(tag: string) { this.tag = tag; }
}
```

A 'static' modifier can be used to declare properties on the constructor function object:

```
class Point
{
    constructor(this.x: number, this.y: number) { }
    getDist() { return Math.sqrt(this.x * this.x + this.y * this.y); }
    setTag(tag: string) { this.tag = tag; }
    static origin = new Point(0, 0);
}
```

Initially, Strada will not support class expressions, but as they are part of the max/min proposal we'll need to eventually. For example, constructs such this should be permitted:

```
function getPointClass(default_x: number, default_y: number) {
    return class
    {
        x = default_x;
        y = default_y;
        getDist() { return Math.sqrt(this.x * this.x + this.y * this.y); }
    }
}
var ZPoint = getPointClass(0, 0);
var NPoint = getPointClass(-1, -1);
var zp = new ZPoint();  // (0, 0)
var np = new NPoint();  // (-1, -1)
```

Anders