Notes - 4/25/2014

Tuesday, April 22, 2014 9:07 AM

Agenda
- Contextual type/BCT change
- ES6 alignment
    - Symbol types
- Abstract classes/methods
- 'protected'

Protected Accessibility

Still the top-most voted issue. More requests this week.

Contextual Type/BCT Change

```
interface Point {
    x: number;
    y: number;
}

var any: any;
// Unexpected error: Cannot convert {}[] to Point[]
var pts: Point[] = [{ x: null, y: any }];

enum Size { Small, Medium, Large }
interface Shirt {
    color: string;
    size?: Size;
}
// Unexpected error: Cannot convert {}[] to Shirt[]
var sizes: Shirt[] = [{ color: 'blue', size: 1 }];

// Poor error message: Type of conditional '{}' must be identical to 'number', 'numbe
r' or 'string'
// Desired error: Cannot convert 'number' to 'string'
var s: string = foo ? 1 : 0;
```

Sounds like all goodness with the new algorithm, even better with Anders optimization.

Destructuring

```
function f() {
        return ["hello", 2];
}  // today returns {}[]

var {a,b} = f();  // not enough to destructure
```

Proposal

arrays can have element types

var x = ["hello", 2];   // would infer {}[string, number]

{}[string, number] ==> [string, number, {}, {}, ...]

x[0] // type string

x = []; // also allowed

We don't check the element types.

Issue:
- how do you assign between x[0] and x[i]? We get two different types, even if i == 0
- Also wouldn't catch x = ["",""]
- What about x[0] = 2? Would break now. Perhaps we never check the element types, we only use it when destructuring

What about having a tuple type?

Possible: never infer [string, number], infer {}[string, number]. If you specify this is a tuple type, we don't let you access beyond that. The [string, number] would be a two element tuple.

Question, how do you name a tuple type? We can't just put it into an interface.

Symbol type examples

Example 1: Assignment compatibility between new symbols

```
function f(): new Symbol {
  return new Symbol();
}

var x = new Symbol();
function g() {
  return x;
}

var a = f();
var b = f();
a = b;  // should fail?

var c = g();
var d = g();
c = d;  // should fail?
```

Example 2: Implementing an interface with a new symbol

```
interface I {
  s: new Symbol;
}
var i: I;

class C implements I {
  s: typeof i.s;
}
```

Example 3: Visibility and new symbols

```
function id2<T>(n: T, m: T) : T { … }

var j = {n: new Symbol() } ;

/* export? */ var k = {s: j.n, t: j.n};
export var m = id2(k.s, k.t);
```

Example 4: New symbols and arrays

```
var a = [new Symbol()];  // what is the type of a?

var b: new Symbol[];
var x = b[0];
x = b[0];  // error?
```

Example 5: A possible hole

```
interface Symbol: { new (s?): new Symbol; }

var q;
function create<T>(ctor: {new (): T}): T {
  var x: T;
  var y: T;
  x == y;
  return q = (q || new ctor());
}

var x = create(Symbol);
var y = create(Symbol);
x = y; // ?
```