Notes - 11/4/2013

Monday, November 4, 2013 9:16 AM

On the agenda for today:

- Inferring type variable without a candidates = error?
  - ACTION ITEM: We need to see how much this would break before we make this change
  - Design decision: Leave where we are, but we should add reasoning about inferring the constraint. In situations where constraints are refer to the type variable (creating a mu type), we would infer {} and fail the constraint, forcing the user to supply the type argument
    - Limiting also cases like: <T, S extends List<T>>
- Do we want all generic types to be covariant? [813197]
  - Design Decision: Modify the constraint checks in contextual signature instantiation to allow co- and contra- variance. This allows "nominal" covariance is true, A<T> assignment compat A<U>, if T assignment compat U

- Issues with multiple inheritance, generics, and recursion [804710]

```
interface A<T>
{
   x : C<T>
}

interface B<T>
{
   x : C<T>
}

interface C<T> extends A<T>, B<T> { }
```

Note: the non-generic case of the above works.
  - Design decision: this appears to just be a compiler bug. Instantiation with C's T should happen as the members from A and B are copied in, which should combine
- Reopen interface w/ call/construct signatures, then reopen and add more call/construct signatures

```
//fileA
interface I {
   (): void;
}

//fileB
interface I {
   (x: number): number;
}
```

- What about signatures that differ only but return type? Arguably, we no longer need to error in this case, we could just pick the first one.
  - We may want to soften this and only give an error if, in the same object literal body, we see two identical members (not including return type)
- --out and file ordering in a VS project
  - _references.ts is the implicit first file to compilation
- .d.ts generation

```
class A {
        private static B;
}

//var x: typeof A;
var x = A;

//.d.ts

class A {
        private static B;
}

//var x: typeof A;
var x: … // try to unroll constructor function, see private, give error

module M {
        var y = {a: 3};
        export var x = y;
}

//.d.ts
module M {
        var y: {a: number};
        export var x: typeof y;  // error?  should unroll
}
```

Rules?
  - If the user wrote a type annotation that we can't emit exactly the same, we give an error
  - In cases of no type annotations…

- When do we use "typeof A"
    - Static side of class
    - Instance side of a module
    - Name is emitted using a crawl back to where the name originates from the current location
- Else we output anonymous type