Notes - 11/14

Wednesday, November 14, 2012 8:35 AM

[In 41/6541 at 10:00 today.] (<u>NOTE</u>: may end up having to move rooms, will send mail before 9:45 if we do)

Agenda:
- Generics update
  - Inference
    - When no type parameters or type arguments explicitly provided - same as today

- Design backlog
  - Overloading on constants
    - No case insensitiv

  - Rubber stamp enum update
  - String property names in interfaces, classes
  - "self type" in classes (see forum thread @ Oct 8 at 3:22 PM)
  - Bringing back expliciy "this" parameter type
  - … for arguments
  - String interpolation
  - Type-only mixins
  - Decorations
  - Update on Fundules, clodules
  - Static initialization (#74)
  - Should best common type produce 'any' in normal (non-infer-any mode)
    - Do we need signaling any?
  - Thinking on how to approach async via generators (#38)
  - SkyDrive feedback – implications for future thinking?
- Others?

```
function foo(day: number): number {}
function foo(day: Weekday): number {}
foo(3)
```

```
// Number based enums
enum Weekday {
  Monday: 0,
  Tuesday: 1
  AnyDay: Weekday.Monday | Weekday.Tuesday
}

// String-based enums
enum Weekday {
  Monday: "Monday!!!!",
  Tuesday: "Tuesday????"
}

// Shorthand for string-based enums, this means the same as above
enum Weekday {
```

```
map<T,U>(items: T[], f(x:T) => U): U[]

var myItems: string[];
var lengths = map(myItems, x => x.length);

// User could write <string, number>, then normal method call

// If not, inference of T,U from actual arguments
```

Algorithm:
- Start with all type parameters unfixed
- For each argument expression:
  - Check whether argument expression requires any type parameters to be fixed
    - If not, intrinsically type the expression
      - Then collect constraints on type parameters
    - If does depend on type parameters ("contextually typing the expression would put a type parameter as part of a parameter type on a funciton expression)
      - Lock down those type parameters that are needed

Q: Does inference report errors?

```
Foo<T>(x: T, y: T): T[];
Var x = [b,c]
Var y = foo(b,c);

var f: <T>(x: T, y: T): T;
f = <U,V>(a: U, b: V) => a;
```

Q: is T 'any' or {}? Previously said {}, but may go to 'any'.
    Start with '{}'

```
  "hello world",
  "Tuesday!!!"
}

// Enums introduce a type
// Also, the raw string value can be assigned to an enum
var x: Weekday = "Monday";

// Enums provide provide access to values using lookup, regardless of underlying
type
var x: Something = Weekday.Monday;
var x: Something = Weekday["Monday"]; // should be an error

// Integer enums can have un-numbered slots.
// Values are auto-incremented from previous numbered slot
enum Weekday {
  Monday: 1,
  Tuesday
} // 1 and 2

// Enum with no values is treated as integer enum starting at 0
enum Weekday {
  Monday,
  Tuesday
} // 0 and 1

// Automatic conversion from underlying type to enum type
enum Method { "GET", "POST", "PUT", "DELETE" }
declare function getMethod(): string;
var method: Method = getMethod(); // Okay to convert from string

// However, it is an error to assign a literal that is not one of allowed
values
var method: Method = "GETT"; // Error - "GETT" is not legal value of Method

document.createElement("Div")

//Enum values are inlined during compilation
var x: Something = Weekday.Monday;
//...becomes...
var x = 0;

// The above means that enum declarations themselves generate no code, they are
erased at compile time
```

From <http://devdiv/sites/bpt/strada/design%20notes/strada%20design%20notes.docx>

- Constant expressions
- Enum value expressions; literal or expr
- Enum type just alias for number/string in assignment compat
- Enum values only unique literals
- Enum is its own map
- Always emit unless 'declare' modifier used