Notes - 10/7: SENT

Friday, October 14, 2011 8:44 AM

```
//
//module Foo {
//  interface A {
//    foo(): number;
//     foo(): {z:number;};
//  }
//
//  var x = (null as A).foo
//  //var w = x();
//
//  interface A1 { clone(): A1; }
//  interface B1 : A1 { clone(): B1; }
//
//
//  // Every member must have a unique signature
//  // The same signature allows specialization
//  // But error if you then don't subtype
//
//
//  var ww = null as A;
//  ww.foo = "hello";
//
//
//  interface X {}
//
//  interface A3 { clone(): A3; clone(s: string): A3; }
//  class C : A { public clone(?s: string) : A3; }
//
//  // const properties - no, and methods should be non-const as well for now
//
//
//  interface I1 { x : A };
//  interface I2 { x : B };
//  interface I : I1, I2 {};
//
//  // RULE: Overloads must all have the same type
//  // RULE: Within a single interface, all members must
//
//  class C() {
//    public foo();
//     public foo(s: string);
//  }
//
//}
//module Foo2 {
//  class A() {
//    public clone(): A;
//     public clone(x: Context): A;
//     public clone(x: number): A;
//     implementation clone(x?:any): A{
//
//     }
//  }
//
//  class B() : A() {
//    implementation clone(x?:any): A {
//
//     }
//  }
//
//  class C() : A() {
//    public clone(): C;
//     public clone(x: Context): C;
//     public clone(x: number): C;
//     implementation clone(x?:any): C {
//
//     }
//  }
//
//
//
//
//
//}


//RULE:

module Foo4 {
//
//  // calls, properties, methods
//  interface A {
//    (): A;
//     (x: string): A;
//     x : A;
//     foo(): A;
//     foo(x: string): A;
//  }
//
//  // calls, properties
//  interface A {
//    (): A;
//     (x: string): A;
//     x : A;
//     foo: { (): A; (x:string): A; };
//  }
//
```

```
//  // call, properties
//  interface A {
//    () | (x: string) : A;
//     x : A;
//     foo: { () | (x: string): A; }
//  }
//
//  interface B {
//    x : B;
//  }
//

  var x = "" + 5;


// We want to avoid having the type of something change the meaning of it
// subtyping determines "treat as"
// all conversions are based on subtyping

// Add workitem

}


module Foo45 {

  interface IArray {
    [x: number]: _element;
  }

  interface IDictionary {
    [x: string]: any;
  }

  interface I1 {
    [x: number]: string;
    [y: string]: string;
  }

}
```

Overloading/i

```
interfac
  foo():
}

interfac
  foo():
}

var b : I
b.foo
```