TypeScript for OO Developers

Or, how to stay cool in front of the millennials
Brexit
Let’s just get it out of the way ...
IS IT SAFE TO COME OUT YET

OR IS EVERYBODY STILL A POLITICIAN???

Make a Meme
James Bond is going to need a Visa for his missions now.
Oh bloody hell, they play cricket as well

KÓPAVOGUR CRICKET CLUB
ICELAND
Agenda

- There is no agenda
- Just me talking TypeScript, javascript and OO
- Trying to follow the slides and examples
  - Constructive criticism will be met with stony silence
  - There will be no talk of my little ponies.
- New questions feature:
  - See the url to ask questions
  - Most popular get moved to the front of the queue ;)
  - Please, be mindful of language and tone - everyone can see what you are writing!
Who are the OO cool kids?

- Never!
- Not yet
- “Researching”
- Sometimes. When I’m forced to.
- Mostly
- I’m a 4GL OO wizard. My OO foo is legendary.
What is TypeScript

It is a superset of Javascript language developed and maintained by Microsoft
YEAH YEAH...WAIT

WHAAAAAT
Microsoft ?

- Yup. Microsoft
- Open source, Cross-platform
- Free
- No runtime libraries
- Developed by Anders Hejlsberg
  - lead architect of C# and creator of Delphi and Turbo Pascal,
- Introduced October 2012
- Adopted by google for Angular2
Getting back to “What is TypeScript ?”

- Static typing
- Class-based OO programming
- supports definition files
  - https://github.com/DefinitelyTyped/DefinitelyTyped
- Compiles down to standard javascript
- Works on client (browser) and server (node.js)
- Strict superset of ECMAScript 2015, which is a superset of ECMAScript 5
I DARE YOU! I DOUBLE-DARE YOU!

SAY ECMAScript ONE MORE GOSH DARN TIME!
ECMAScript === javascript

- See what I did there?
- TypeScript code (.ts) is transpiled
- Creates new .js code
- Compiler has watchers (compile on save)
- JS build tools (grunt / gulp etc) have ts support
4GL vs JS challenges

- Equals
- Typesafe
- Classes
- Variable declaration
Example of js gotcha

```javascript
var x = 1;
var y = 2;

if (x == y) {
  console.log("yay");
} else {
  console.log("boo");
}
```
Quick - some OO (TS & 4GL) advantages

- Type-safe
  - No spelling errors
- Code Reuse and Recycling
- Encapsulation
- Makes you sound cool
  - "Injection", "factory", "instantiation", "inheritance"
Code formatting

- Code formatting in this presentation will offend
- Both 4GL and JS coders
- Tough
- I have to work within the constraints of a slide ;}
There are two types of people.

Programmers will know.
Why A TypeScript and 4GLOO presentation?

- Typescript classes very similar to 4GL OO classes
- Very similar functionality
- Helps with transition to javascript
- Opens new possibilities for the cool 4GL OO developers
- Why not? ;)
class Greeter:
    def public property greeting as char no-undo get . set .
    constructor(message as char):
        assign this-object:greeting = message;
    end constructor.

    method public char greet():
        return "Hello, " + this-object:greeting.
    end method.
end class.
class Greeter{
  greeting: string;
  constructor(message: string){
    this.greeting = message;
  }
  greet(){
    return "Hello, " + this.greeting;
  }
}
OO 4GL vs TypeScript: abstracts

- No, not abstracts. Abstracts ;)
- We’ll get to real abstracts later
Files: 4GL

- each class defined in a single .cls file
- Compiles to a single .r
- Folder structure is the “package” or namespace
  - Shapes/Polygons/Triangle.cls gives Shapes.Polygons.Triangle class
  - Shapes/Polygons/Square.cls gives Shapes.Polygons.Square class
- **Using** statement allows for shortcuts
  - Using Shapes.*
  - A = new Polygons.Square()
Files: TypeScript

- One or more classes defined per .ts file
- Namespaces defined by , well, namespace ;)

```typescript
namespace Shapes {
    export namespace Polygons {
        export class Triangle { }
        export class Square { }
    }
}
```
• import statement used for shortcuts

```typescript
import polygons = Shapes.Polygons;
let sq = new polygons.Square();
```
Compiling

- **4GL:**
  - compile Shapes/Polygons/Square.cls save [options]
  - Produces single .r file per .cls
- **TS:**
  - tsc [options] shapes.ts
  - Produces single .js file per .ts
- **Both have compiler tools / utilities**
  - Eclipse compile on save (4GL & TS)
  - Grunt / gulp file watchers (TS)
- **Both have full IDE support (colour highlighting, syntax etc)**
class Greeter{
    greeting: string;
    constructor(message: string){
        this.greeting = message;
    }

    greet(){
        return "Hello, " + this.greeting;
    }
}

Compiler options

https://www.typescriptlang.org/docs/handbook/compiler-options.html
Classes

- Defining a class is very similar to the 4GL

```cpp
class <name> {
    public <item>: <datatype>
    public constructor
    public method ..
}
```
constructors

- JS does not have function overloading
- As a TS constructor is transpiled to a function, overloading is not possible
- Advanced workarounds
export class ShoppingListItem implements IShoppingListItem {
    constructor(item: IShoppingListItem);
    constructor(name: string, amount: number);

    constructor(nameOrItem: string | IShoppingListItem, amount?: number) {
        if (typeof nameOrItem === "object") {
        } else {
            if (typeof nameOrItem === "string" && typeof amount === "number") {
            }
        }
    }
}
Potential problems with such an approach

- Typescript can check constructor parameters
- Javascript cannot
  - Pass in `{foo: "bar"}` as a parameter, it will be accepted
- Code becomes ugly
  - Can lead to reactions like this
Destructors. This is simple.

- There is no such thing in typescript.
ONE DOES NOT SIMPLY
HAVE DESTRUCTORS
Properties

- Same as 4GL
  - Private
  - Public
  - Protected

```java
class Person {
    private title: string;
    public firstName: string;
    protected name: string;
}
```
class Employee {
    private _fullName: string;

    get fullName(): string { return this._fullName; }

    set fullName(newName: string) {
        this._fullName = newName;
    }
}
Methods

● Same as 4GL
  ○ Private
  ○ Public
  ○ Protected

```java
class Person {
    private move() {};
    public anotherMove() {};
    protected myLittlePony() {};
}
```
Statics

- Static properties
- Static methods
- Static classes can be created by throwing an error in the constructor
  - RUNTIME only though ...

```javascript
class MyClass {
  constructor() { throw new Error("Cannot new class"); }
  static myProp = "Hello";
  static doSomething() { return "World"; }
}
```
Inheritance

- Classes can extend other classes (inherit)
- Exactly the same as 4GL
- Multiple levels
  - More than 3 is bad, no matter what language you use ;)


class Animal {
    name: string;
    constructor(theName: string) { this.name = theName; }
    move(dist: number = 0) { console.log(`moved ${dist}m`); }
}

class Snake extends Animal {
    constructor(name: string) { super(name); }
    move(dist = 5) { super.move(distance); }
}

class Horse extends Animal {
    constructor(name: string) { super(name); }
    move(dist = 45) { super.move(dist); }
}
Interfaces

- Specifies a set of method prototypes and properties
- No default implementation methods or properties
- Allows you to build different classes that conform to an API
- Each class that uses an Interface must implement all methods and properties defined in the interface
- Convention is that interface files start with an I
interface IClockInterface {
    currentTime: Date;
    setTime(d: Date);
}

class Clock implements IClockInterface {
    currentTime: Date;
    setTime(d: Date) {
        this.currentTime = d;
    }
    constructor(h: number, m: number) {
    }
}
Polymorphism

- More complicated in TypeScript
- Create overload methods using similar techniques to constructors
- Override a method possible by not calling super()
  - See samples/Inheritance.ts
Abstracts

- See, I told you we’d get here
- Very similar to 4GL
- Abstracts are classes than cannot be instantiated by themselves
- Need to inherit an abstract class
- All appropriate properties and methods are available to sub-class
abstract class Animal {

    abstract makeSound(): void;

    move(): void {
        console.log("roaming the earth...");
    }
}
Events: again, this is simple

- There are no class events
- You can use standard event emitters
- Pub / sub
- Etc
ONE DOES NOT SIMPLY
HAVE CLASS EVENTS
“Friends, time is not just short. We have run out of time. This is our historical moment. Let us not disappoint. The stakes are simply too high. Now is not the time for small steps. Now is the time for boldness. Now is the time to leap.” —Naomi Klein
Converting classes : Greeter.cls
class Greeter{
    public greeting: string;

    public constructor (initMessage: string){
        this.greeting = initMessage;
    }

    public greet(){
        return "Hello, " + this.object.greeting.
    }

    end method.

    end class.
}