Working with Generics and Interfaces

Chris B. Behrens
Senior Software Architect

@chrisbbehrens

Really Understanding Interfaces

On the day you need to drive a screw, you will understand the screwdriver

A contract to implement certain members

This is also true of an abstract class

An interface guarantees that certain methods or properties are in place so that something else can call them in order to get its work done.

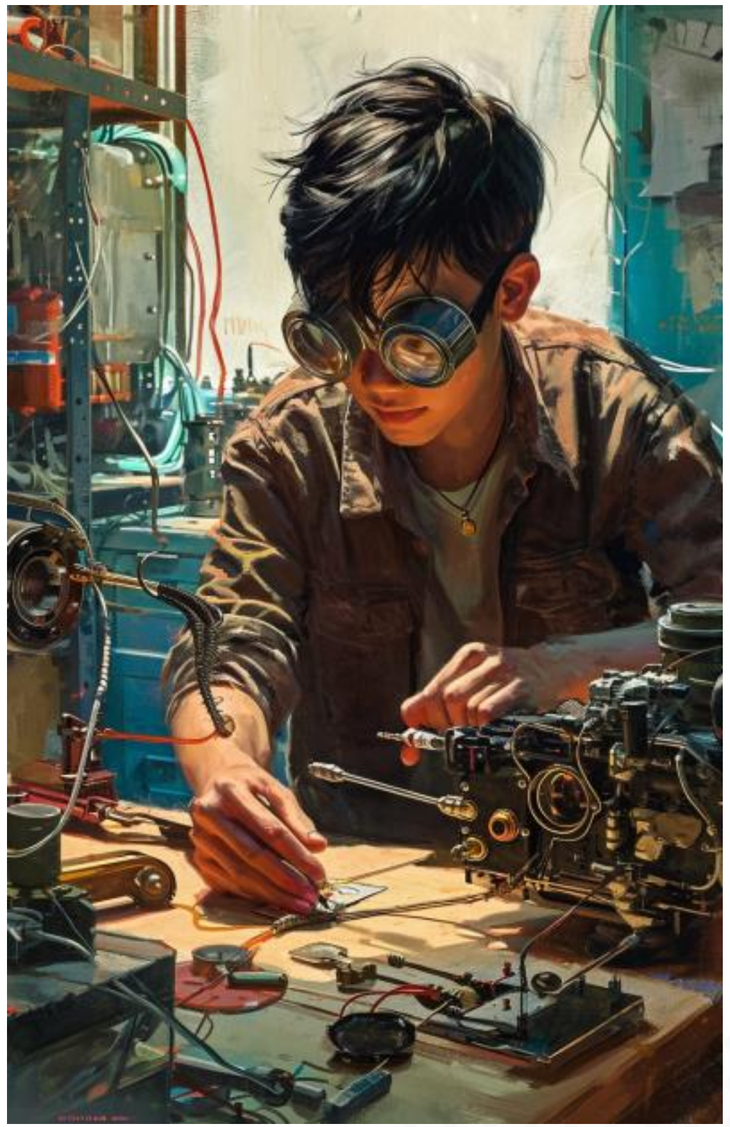
ILogger

A commonly used interface

Used by Globomantics products

Two methods, LogEvent, and LogException

We're deferring the details of what "logging" means



Ship with a text logger, XML, JSON,
and maybe db

I can try to predict what users might need...

But there will always be needs
you cannot predict

Better to let them roll their own

"Create observable evidence"

Logged, but...where?

You might transport the logs
to a central server

Or just broadcast directly to the EventHub

Demo: Implementing a Third-Party TS Interface

Our ILogger interface

A concrete implementation of the ILogger interface

Our EventHubLogger

How all this works together

What Generics Are for

Getting Away from “Any”

```
function write(contents: string){
    //do stuff
}

write(1234);

function writeNumber(contents: number){
    // do stuff
}

function writeBool(contents: Boolean)
function writeDate(contents: Date)
function write(contents: any)
```

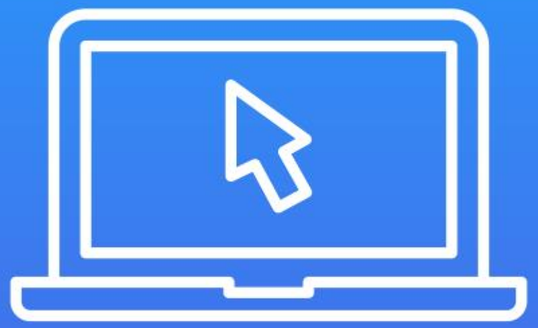
A Simple Generic Argument

```
function write<Type>(arg: Type){
     // write type-specific stuff
}

write<string>(“Chris B. Behrens”);
write<number>(12345);
write<Boolean>(false);
```

You use generics when you want the benefits of typing, but you want to defer the decision about the typing to the other developer.

Demo: Implementing a Generic in TS



A generic key value pair in an interface

Consume it in a couple of different ways

Migrating a fixed type to a generic

A use case for generic classes

Consume it in several ways

Generic Constraints

Key-Value Pair is a primary use case

But you don't get far before you need to get work done

ICommand

```
export interface ICommand{
    execute();
}

//StandUpCommand, SitDownCommand, WalkAcrosstheRoomCommand

class invoker{
    executeCommand(command: ICommand){
        command.execute();
    }
}
```

Duck Typing and Generics

Something else with an execute(*something*) method

Not strictly implementing the ICommand interface

Duck-typing

Having the same methods

TypeScript can make it happen

GenericInvoker.ts

```
export class Invoker{
    executeCommand<Type extends ICommand>(command: Type){
        command.execute();
    }
}
```

Summary

- Interfaces
- A handful of demonstrations
- Generics
- The deferral of type resolution
- Several different ways to make it work