Generics

PROGRAMMING WITH TYPESCRIPT

Objectives

- To understand how to use Generics with:
  - Functions
  - Interfaces
  - Classes
  - Constraints

2

Generics

- Generics are a tool we can use to create reusable components that work over a variety of data types, rather than a single one.
- This enables us to be DRY and strongly typed

3

Generic Functions

- Using the special type variable identified by the angle brackets we can create a generic function that can accept (and capture!) any type
- This is called a type variable
- The function (right) is generic as it works across a range of types, and unlike using the any type, it retains precision.
- We then call the function stating the type or allowing inference

```
function logVehicle<T>(vehicle: T): T {
  console.log(vehicle);
  return vehicle;
}
```

```
let loggedVehicle = logVehicle<Car>(myCar); //specified
let loggedVehicle = logVehicle(myCar); //inferred
```

4

Generic Functions

- Bear in mind that T can be any type, so we have to treat it as such

```
function logVehicle<T>(vehicle: T): T {
  console.log(vehicle.length);
  return vehicle;
}
```

- Why does the above Error?

5

Answer: because not all types have a member called “length”

Generic Interfaces

- We can write generic interfaces, often to describe objects that will be generics themselves

```
class VehicleStorage<T> {
  vehicles: T[];
  park: (arg: T): T[];
}
```

6

Generic Classes

- Similar in shape to generic interfaces, utilising angle brackets following the name of the class
- Generic classes are only generic on their instance side, so static members cannot use the class's type parameter

```
class VehicleGarage<T> {
  vehicles: T[] = [];
  constructor(public space: number = 5) {}
}
```

7

Generic Constraints

- We can constrain Generics to only accept types within certain parameters
- Whereus until now we've had to work with our generics as if they were accepting any types, we can now work with them in certain conditions

```
interface RoadVehicle {
  height: number;
  weight: number;
}

class VehicleGarage<T implements RoadVehicle>
{
  vehicles: T[] = [];
  constructor(public space: number = 5) {}
}
```

8

Objectives

- To understand how to use Generics with:
  - Functions
  - Interfaces
  - Classes
  - Constraints

9

QuickLab 5 – TypeScript Generics

- Experiment with Generics – classes, constraints and functions

10

Hackathon Part 1 – Form Validation using TypeScript

- In this part of the Hackathon, you will build on a partially developed solution (whether that be your previous iteration or the provided starting point) for QA Cinemas' website by adding validation to the form, writing type-safe JavaScript using TypeScript. All the necessary tools, knowledge and techniques have been covered in the course so far.
- This part of the Hackathon is intended to help you develop your skills and knowledge to be able to use type-safe JavaScript via TypeScript to validate a 'Sign-Up' form for users of the QA Cinemas website before this is sent to be held on the server.

11

CONTINUED 11