TypeScript

Intro to TypeScript

- TypeScript is a superset of JavaScript with added features Java Devs will recongnize

TS Data Types

(primative types are lower case)

- number - equivalent to a Java double, this is used to decalare any type of number
- boolean - same as Java
- string - same as Java
- symbol - Very rarely used
- null - variable has not been given a valid value
- undefined - variable has not been initalized
- Array - Similar to Java List, they can be any size and grow and shrink over time, and you can use generics as you would in Java

```
  exploringArrays() {
    const myArray1 = new Array<number>();
    const myArray2: number[] = [1, 2, 3, 4];
  }
```

- Tuple - Array of fixed size of any object
- any - Equivanlent to Object type in Java
- object - Used to represent any non-primative data types
- void - Similar to Java in that it means a function can return nothing, but a variable can also be set to void

Take the below code for example

```
export class AppComponent {
  title = 'ExploringTypeScript';


  readonly myNumber: number;
  someMethod(): void {
    let anotherNumber;
    const aThirdNumber = 1;

    anotherNumber = 2;
  }
}
```

- Variables declared at the method level must have the `let` or `const` keyword
- As in JS you don't need to declare the type, but once a variable is assigned to a type it cannot be changed
- By default all class-level variables are mutiable, but this can be changed with the `readonly` keyword

Working with Arrays

- `myArray.slice(<start index>, <end index>)` returns an array from start index up to but not including end index
- `myArray.splice(<start index>, <number of elements to remove>`)` remove elements in array
- `myArray.pop()` removes and returns the last element from the list
- `myArray.push()` add element to the end of the list

Loops

```
for (let i = 0; i < 10; i++) {
	console.log(i);
}

for (let next of myArrray){
	console.log(next);
}
```

- Be careful not to use `in` instead of `of` above, that will fetch the indices rather than the values
- In the above, you could change `next` to the `const` modifier, as it never changes within the loop. But `i` could not be changed to `const` as the values changes throughout the loop.

Classes and Objects

Because TypeScript is compiled to JavaScript classes don't actaully exist at runtime.

- JavaScript's concept of an Object is a set of key-value pairs. `let myObject = {firstname: 'Bob', age: 20}`
- These also exist in TypeScript, but classes are far more powerful.
- Also, unless there is some sort of biussness logic involved, getters and setters are pointless because anyone can access them. Unlike in Java, just make your variables public
- You also cannot have multiple constructors, even with different parameters. Instead you can have Optional parameters
- Also unlike Java, multiple classes can exist in one file.

```
export class Book {
  title: string;
  author: string;
  price: number;
  readonly id: number = 1;

  constructor(author: string, title?: string) {
    this.author = author;
    if (title) {
      this.title = title;
    }
  }
}
```

- Enums also exist, and they are treated as arrays
  - Values 0 to n are automatically assigned to each entry, but this can be changed
  - Also the looping behavior changes if you set it to custom values

```
enum SubjectArea {
  ART, HISTORY, SCIENCE, LITURATURE
}

printArrays(): void {
    for (const subject in SubjectArea) {
      if (true) {// the if statement just surpresses a warning
        console.log(subject);
      }
    } // output: 0, 1, 2, 3, ART, HISTORY, SCIENCE, LITURATURE

    const enumArray = Object.keys(SubjectArea);
    for (const value of enumArray.slice(enumArray.length / 2)) {
      console.log(value);
    } // output: ART, HISTORY, SCIENCE, LITURATURE
}

export enum SubjectArea {
  ART = 'Arts and Shit', HISTORY = 'History', SCIENCE = 'Science and Math', LITURATURE = 'English'
}

printArrays(): void {
let label;
    for (const subject in SubjectArea) {
        console.log(subject);
        console.log(SubjectArea[subject]);
        if (SubjectArea[subject] === 'History') {
          label = subject;
        }
    }

    let label2 = Object.keys(SubjectArea).find( it => SubjectArea[it] === 'History' );
}
```

Methods and Functions

`methodName(paramName: paramType): ReturnType`

- TS methods are public by default, use the `private` attribute to change that
- If a class contains methods, during runtime they will not be availble from the console as all instances of classes are converted to generic JS objects. You could create a function to convert the object to JSON, for ex:

```
export class User {
  id: number;
  name: string;
  getRole(): string {
    return 'standard';
  }

  static fromHttp(user: User): User {
    const newUser = new User();
    newUser.id = user.id;
    newUser.name = user.name;
    return newUser;
  }
}
```

```
    this.dataService.getUser(13).subscribe(
      next => {
        console.log(next);
        console.log(typeof next);
        let user: User = next;
        console.log(user)
        console.log(typeof user)
        let user2: User = next as User;
        console.log(user2)
        console.log(typeof user2)
        let user3: User = User.fromHttp(next);
        console.log(user3);
        console.log(typeof user3)
□□// this will only work for users created from the created method
        console.log(user3.getRole())
      });
```

Output:

```
{ id: 13, name: "matt" }
object
{ id: 13, name: "matt" }
object
{ id: 13, name: "matt" }
object
{ id: 13, name: "matt" }
object
standard
```

Reminder: JavaScript SUCKS

JavaScript is not an 'opinionated' langauge, meaning there are a million ways to do the same fucking thing with no reccomendation on how to optimize. Ex:

```
console.log(`To buy this book it will cost: ${myBook.priceWithTax(.2)} dollars`);
console.log('To buy this book it will cost: ' + myBook.priceWithTax(.2) + ' dollars');
console.log('To buy this book it will cost: ', myBook.priceWithTax(.2), ' dollars');
```

All do the same thing. Notice the backticks instead of quotes in the first.

- Actaully the last console statement above would print an Object in JSON form. If you're ever debugging and keep seeing Object object, try switching to commas. Also filtering can be done in different styles:

```
  const oddNumbers = numbers.filter(
    num => {
      return num % 2 === 1;
    }
  );

  const evenNumbers = numbers.filter( num => num % 2 === 0 );
```

But by far the most mind-bogglingly stupid feature of JS is eqaulity. == Abstract eqaulity === Strict eqaulity The short story is always use Strict eqaulity. As abstract eqaulity will attempt to cast parameters to the same type before comparision.

Revision #1
Created 16 April 2022 23:35:30 by Elkip
Updated 16 April 2022 23:40:55 by Elkip