Interface

- An Interface is defined using the `interface` keyword
- Interfaces are used only during compilation time to check types
- By convention, interface definitions start with an `I`, e.g. : `IPoint`
- Interfaces are used in classical object oriented programming as a design tool
- Interfaces don't contain implementations
- They provide definitions only
- When an object implements an interface, it must adhere to the contract defined by the interface
- An interface defines what properties and methods an object must implement
- If an object implements an interface, it must adhere to the contract. If it doesn't the compiler will let us know.
- Interfaces also define custom types

Basic Interface

Below is an example of an Interface that defines two properties and three methods that implementers should provide implementations for:

```
interface IMyInterface {
  // some properties
  id: number;
  name: string;

  // some methods
  method(): void;
  methodWithReturnVal():number;
  sum(nums: number[]):number;
}
```

Using the interface above we can create an object that adheres to the interface:

```
let myObj: IMyInterface = {
  id: 2,
  name: 'some name',

  method() { console.log('hello'); },
  methodWithReturnVal () { return 2; },
  sum(numbers) {
    return numbers.reduce( (a,b) => { return a + b } );
  }
};
```

Notice that we had to provide values to **all** the properties defined by the Interface, and the implementations for **all** the methods defined by the Interface.

And then of course you can use your object methods to perform operations:

```
let sum = myObj.sum([1,2,3,4,5]); // -> 15
```

Some Angular Interfaces

Angular uses interfaces all over the place. The interfaces that are used very often are the *LifeCycle Hooks*.

LifeCycle Interfaces

```
export interface OnChanges {
  ngOnChanges(changes: {
    [key: string]: SimpleChange
  });
}

export interface OnInit {
  ngOnInit();
}

export interface OnDestroy {
  ngOnDestroy();
}
```