TypeScript Basics

What is TypeScript?

- Language developed by Microsoft in 2012
  - Free and open-source

www.typescriptlang.org

- Provides static typing support to JavaScript
  - Helps with IDE support: code completion and debugging

- Add support for object-oriented programming
  - Classes, objects, inheritance, interfaces, etc …

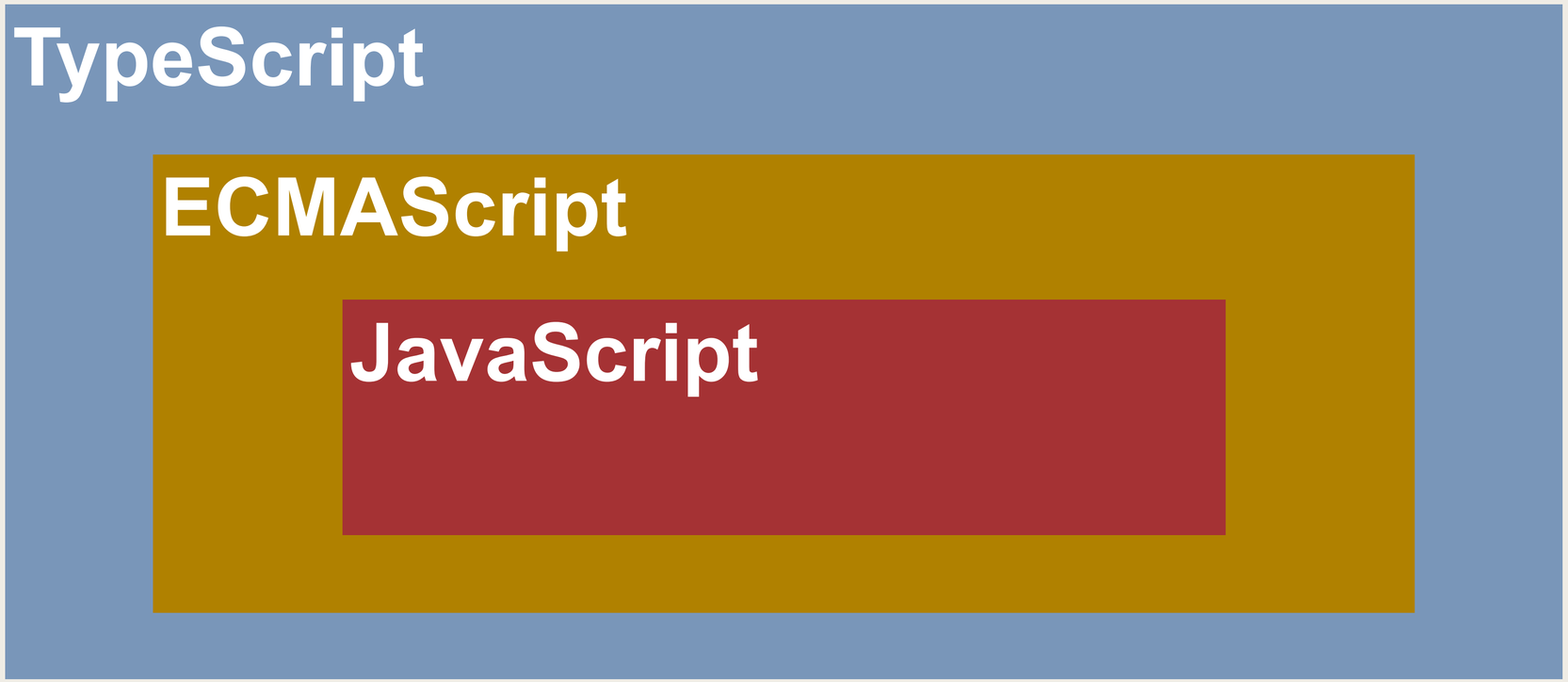
Angular Development

- For Angular development, we can develop using various languages
  - **JavaScript**: extremely popular programming language
  - **ECMAScript:** standardized version of JavaScript (ES6, ES9, …)
  - **TypeScript:** adds optional types to JavaScript
  - *Other languages such as **Dart**, etc …*

- **TypeScript** is the most popular language for Angular development

Relationships

- TypeScript is a superset of JavaScript and ECMAScript



TypeScript

- **FAQ: Why do most Angular developers use TypeScript?**

- Strongly-typed language with compile time checking and IDE support
- Increased developer productivity and efficiency

- The Angular framework is internally developed using TypeScript
- Docs, online blogs and tutorials use TypeScript for coding examples

Practical Results

- Introduction to TypeScript development
- Not an A to Z reference
- For complete reference, see TypeScript Documentation

www.typescriptlang.org

Development Process

Step-By-Step

1. Create TypeScript code

2. Compile the code

3. Run the code

Step 1: Create the TypeScript code

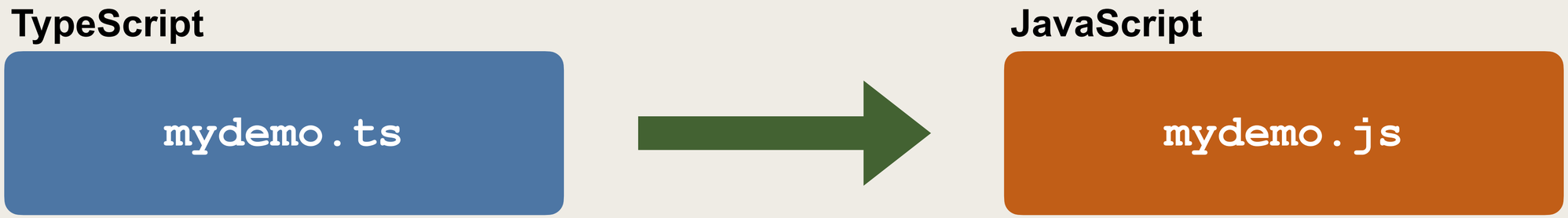
- TypeScript files have the **.ts** extension

File: mydemo.ts

```
console.log("Hello World!");
```

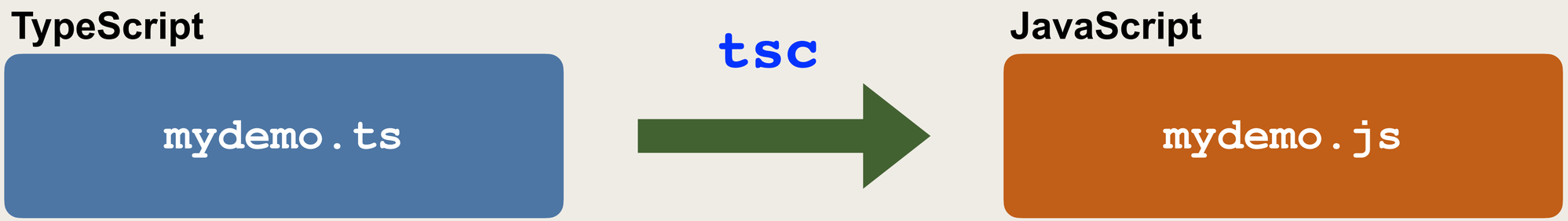
Step 2: Compile the Code

- Web browsers do not understand TypeScript natively
- Have to convert TypeScript code to JavaScript code
- This is known as **"transpiling"**



Step 2: Compile the Code (cont)

- **"Transpiling"** is accomplished with the **`tsc`** command




```
C:\> tsc mydemo.ts
```

Step 3: Run the code

- To run the JavaScript code, we use the **`node`** command
- Run the **generated JavaScript code** (**`.js`** file)

```
C:\> node mydemo.js

Hello World!
```

```
console.log("Hello World!");
```

The Compiler is Your Friend

- The compiler / IDE can find errors earlier at compilation time

```
console.LOGSTUFF("Hello World!");
```

Compile code using: tsc

```
C:\> tsc mydemo.ts

myhello.ts:1:9 - error TS2339: Property 'LOGSTUFF' does not exist on type 'Console'.

 console.LOGSTUFF("Hello World!!");
         ~~~~~~~~

Found 1 error.
```

Compilation error … much better

TypeScript Variables

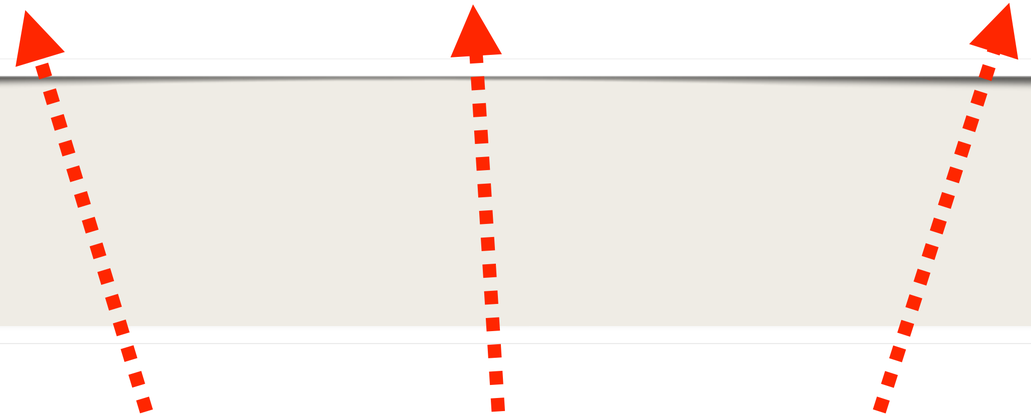
Basic Types

Type	Description
`boolean`	**true/false values**
`number`	**Supports integer and floating point numbers**
`string`	**Text data. Enclosed in single or double quotes**
`any`	**Supports "any" datatype assignment**
Others …	**See details at www.typescriptlang.org**

Define Variables

Syntax

```
let <variableName>: <type> = <initial value>;
```



Example

```
let found: boolean = true;
```

Examples

```
let found: boolean = true;

let grade: number = 88.6;

let firstName: string = "Anup";

let lastName: string = 'Kumar';
```

Double-quotes

Single-quotes

Examples

```
let found: boolean = true;

let grade: number = 88.6;

let firstName: string = "Anup";

let lastName: string = 'Kumar';
```

true or false

73
64.5
100

-quotes

Single-quotes

TypeScript: "let" keyword

- We are using the new TypeScript **`let`** keyword for variable declarations
  - As opposed to using traditional JavaScript **`var`** keyword

- The JavaScript **`var`** keyword had a number of gotchas and pitfalls
  - Scoping, capturing, shadowing etc

- The new TypeScript **`let`** keyword helps to eliminate those issues

TypeScript is Strongly Typed

```
let found: boolean = true;
let grade: number = 88.6;
let firstName: string = "Anup";
let lastName: string = 'Kumar';


// this is okay ... we can assign to different values

found = false;
grade = 99;
firstName = 'Eric';
lastName = 'Noh';
```

This is ok

TypeScript is Strongly Typed

```
let found: boolean = true;
let grade: number = 88.6;
let firstName: string = "Anup";
let lastName: string = 'Kumar';
```

```
// this will generate compilation errors ...

found = 0;
grade = "A";
firstName = false;
lastName = 2099;
```

Type mismatch

Type: any

```
let myData: any = 50.0;

// we can assign different values of any type

myData = false;
myData = 'Eric';
myData = 19;
```

This is ok
But be careful …
you lose type-safety

Displaying Output

File: sample-types.ts

```
let found: boolean = true;
let grade: number = 88.6;
let firstName: string = "Anup";
let lastName: string = 'Kumar';

console.log(found);
console.log("The grade is " + grade);
console.log("Hi " + firstName + " " + lastName);
```

```
true
The grade is 88.6
Hi Anup Kumar
```

Run the App

File: sample-types.ts

```
let found: boolean = true;
let grade: number = 88.6;
let firstName: string = "Anup";
let lastName: string = 'Kumar';

console.log(found);
console.log("The grade is " + grade);

console.log("Hi " + firstName + " " + lastName);
```

Compile code using: tsc

Remember, tsc generates a .js file

```
C:\> tsc sample-types.ts

C:\> node sample-types.js
true
The grade is 88.6
Hi Anup Kumar
```

Run code using: node

Run the .js file

Template Strings

```
let firstName: string = "Anup";
let lastName: string = 'Kumar';

console.log("Hi " + firstName + " " + lastName);
```

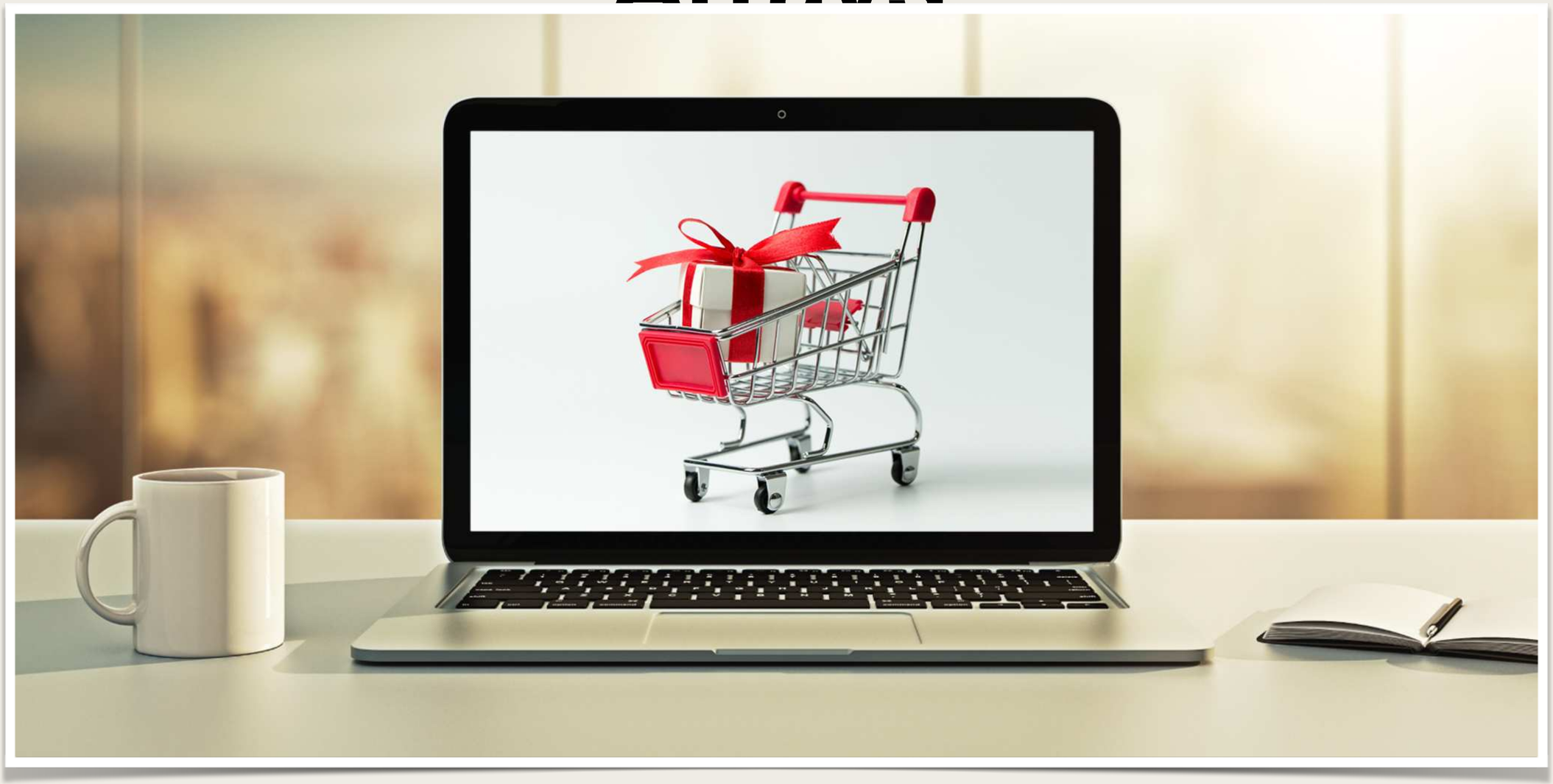
Concatenation could become clunky for long strings

```
console.log(`Hi ${firstName} ${lastName}`);
```

Use backticks: `
Reference variables with ${..}

Useful for long strings with a lot of concatenation

TypeScript Loops and Arrays



For Loops

Compile code using: tsc

File: loops.ts

```
for (let i=0; i < 5; i++) {
    console.log(i);
}
```

```
C:\> tsc loops.ts

C:\> node loops.js
0
1
2
```

Run code using: node

For Loop - Array of numbers

File: reviews.ts

```
let reviews: number[] = [5, 5, 4.5, 1, 3];

for (let i=0; i < reviews.length; i++) {
    console.log(reviews[i]);
}
```

Declare an array

Index into the array

```
C:\> tsc reviews.ts

C:\> node reviews.js
5
5
4.5
1
3
```

For Loops - Compute Average

File: reviews.ts

```
let reviews: number[] = [5, 5, 4.5, 1, 3];

let total: number = 0;

for (let i=0; i < reviews.length; i++) {
    console.log(reviews[i]);
    total += reviews[i];
}

let average: number = total / reviews.length;

console.log("Review average = " + average);
```

Same as: total = total + reviews[i];

```
5
5
4.5
1
3
Review average = 3.7
```

Arrays

File: sports.ts

```
let sportsOne: string[] = ["Golf", "Cricket", "Tennis", "Swimming"];

for (let i = 0; i < sportsOne.length; i++) {

    console.log(sportsOne[i]);
}
```

Index into the array

```
C:\> tsc sports.ts

C:\> node sports.js
Golf
Cricket
Tennis
Swimming
```

Simplified For Loop

File: sports.ts

```
let sportsOne: string[] = ["Golf", "Cricket", "Tennis", "Swimming"];

for (let tempSport of sportsOne) {

    console.log(tempSport);
}
```

Current array element

```
C:\> tsc sports.ts

C:\> node sports.js
Golf
Cricket
Tennis
Swimming
```

Conditionals

File: sports.ts

```
let sportsOne: string[] = ["Golf", "Cricket", "Tenni

for (let tempSport of sportsOne) {

    if (tempSport == "Cricket") {
        console.log(tempSport + " << My Favorite!");
    }
    else {
        console.log(tempSport);
    }

}
```

Conditional

```
C:\> tsc sports.ts

C:\> node sports.js
Golf
Cricket << My Favorite!
Tennis
Swimming
```

Growable Arrays

Arrays in TypeScript are always growable / dynamic

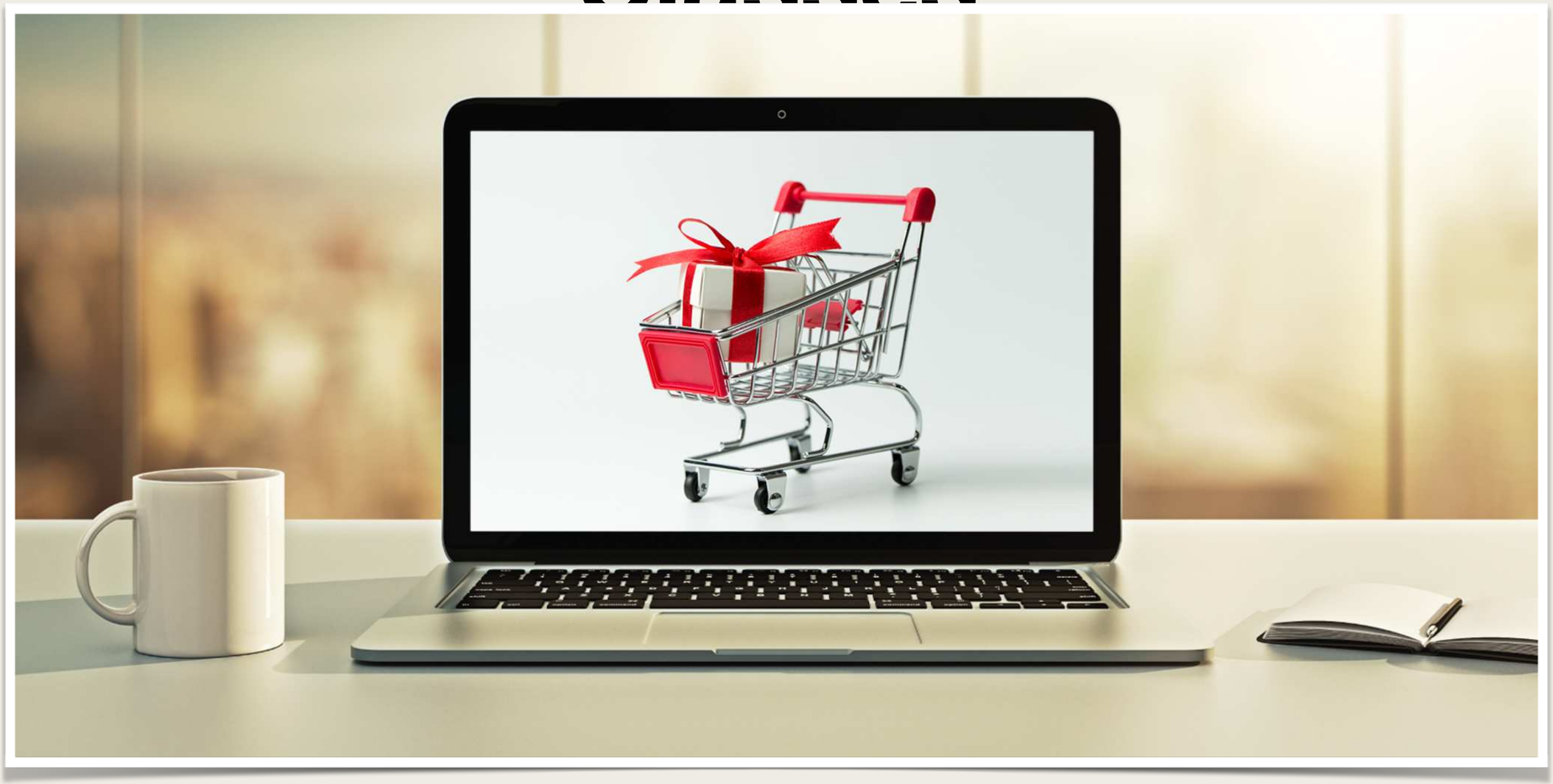
File: growable-arrays.ts

```
let sportsTwo: string[] = ["Golf", "Cricket", "Tennis"];

sportsTwo.push("Baseball");
sportsTwo.push("Futbol");

for (let tempSport of sportsTwo) {
    console.log(tempSport);
}
```

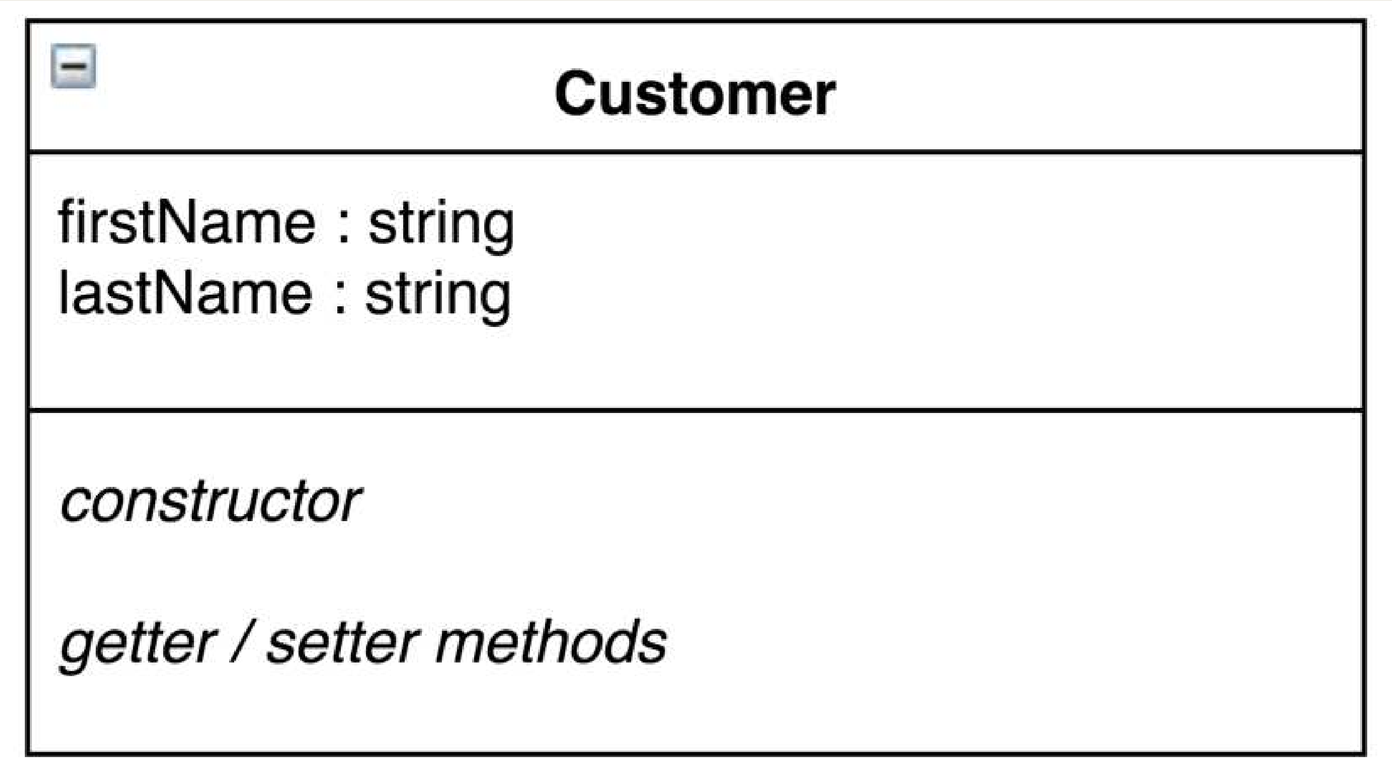
Add elements

```
C:\> tsc growable-arrays.ts

C:\> node growable-arrays.js
Golf
Cricket
Tennis
Baseball
Futbol
```

TypeScript - Creating Classes



Customer Class



Basic Structure

File: Customer.ts

```
class Customer {

    // properties

    // constructors

    // getter / setter methods

}
```

Can use any file name: *.ts

mydemo.ts

Properties

File: Customer.ts

```
class Customer {

    // properties
    firstName: string;
    lastName: string;


}
```

Properties are public by default

We'll cover access modifiers shortly

Construct an Instance

File: Customer.ts

```
class Customer {

    // properties
    firstName: string;
    lastName: string;

}

// now let's use it
let myCustomer = new Customer();

myCustomer.firstName = "Martin";
myCustomer.lastName = "Dixon";

console.log(myCustomer.firstName);
console.log(myCustomer.lastName);
```

Construct an instance using the "new" keyword

```
C:\> tsc Customer.ts

C:\> node Customer.js
Martin
Dixon
```

Create a Constructor

File: Customer.ts

```
class Customer {

    firstName: string;
    lastName: string;

    constructor(theFirst: string, theLast: string) {
        this.firstName = theFirst;
        this.lastName = theLast;
    }
}
```

Use the "constructor" keyword

Must use "this" to refer to properties defined in this class

Construct an Instance

File: Customer.ts

```
class Customer {

    firstName: string;
    lastName: string;

    constructor(theFirst: string, theLast: string) {
        this.firstName = theFirst;
        this.lastName = theLast;
    }
}

// now let's use it
let myCustomer = new Customer("Martin", "Dixon");

console.log(myCustomer.firstName);
console.log(myCustomer.lastName);
```

Construct an instance using our new constructor

```
C:\> tsc Customer.ts

C:\> node Customer.js
Martin
Dixon
```

Access Modifiers

Modifier	Definition
`public`	Property is accessible to all classes (default modifier)
`protected`	Property is only accessible in current class and subclasses
`private`	Property is only accessible in current class

Mark the properties as "private"

File: Customer.ts

```
class Customer {

    private firstName: string;
    private lastName: string;

    constructor(theFirst: string, theLast: string) {
        this.firstName = theFirst;
        this.lastName = theLast;
    }
}

// now let'
let myCusto

(property) Customer.firstName: string
Property 'firstName' is private and only accessible within class 'Customer'. ts(2341)
Peek Problem  No quick fixes available

myCustomer.firstName = "Susan";
myCustomer.lastName = "Public";

console.log(myCustomer.firstName);
console.log(myCustomer.lastName);
```

Compilation error

Compiling the Code

```
C:\> tsc Customer.ts

Customer.ts:16:12 - error TS2341: Property 'firstName' is private and only
accessible within class 'Customer'.

16 myCustomer.firstName = "Susan";
…
…
…
```

```
myCustomer.firstName = "Susan";
myCustomer.lastName = "Public";
```

```
class Customer {

    private firstName: string;
    private lastName: string;
```

Be Careful!!!!

- Even though there are compilation errors …
- The TypeScript compiler will STILL generates a .js file! Yikes!!

```
C:\> tsc Customer.ts

Customer.ts:16:12 - er                                  is private and only
accessible within clas
…

C:\> dir

Customer.js     Customer.ts
```

What???

…. and the code will still run!

```
C:\> tsc Customer.ts

Customer.ts:16:12 - error TS2341: Property 'firstName' is private and only
accessible within class 'Customer'.
…

C:\> dir

Customer.js     Customer.ts

C:\> node Customer.js
Susan
Public
```

What???

```
// now let's use it
let myCustomer = new Customer("Martin", "Dixon");

myCustomer.firstName = "Susan";
myCustomer.lastName = "Public";

console.log(myCustomer.firstName);
console.log(myCustomer.lastName);
```

We Can Prevent This!

Remove previous .js file

```
C:\> del Customer.js

C:\> tsc --noEmitOnError Customer.ts

Customer.ts:16:12 - error TS2341: Property 'firstName' is private and only
accessible within class 'Customer'.
…

C:\> dir

Customer.ts
```

We Can Prevent This!

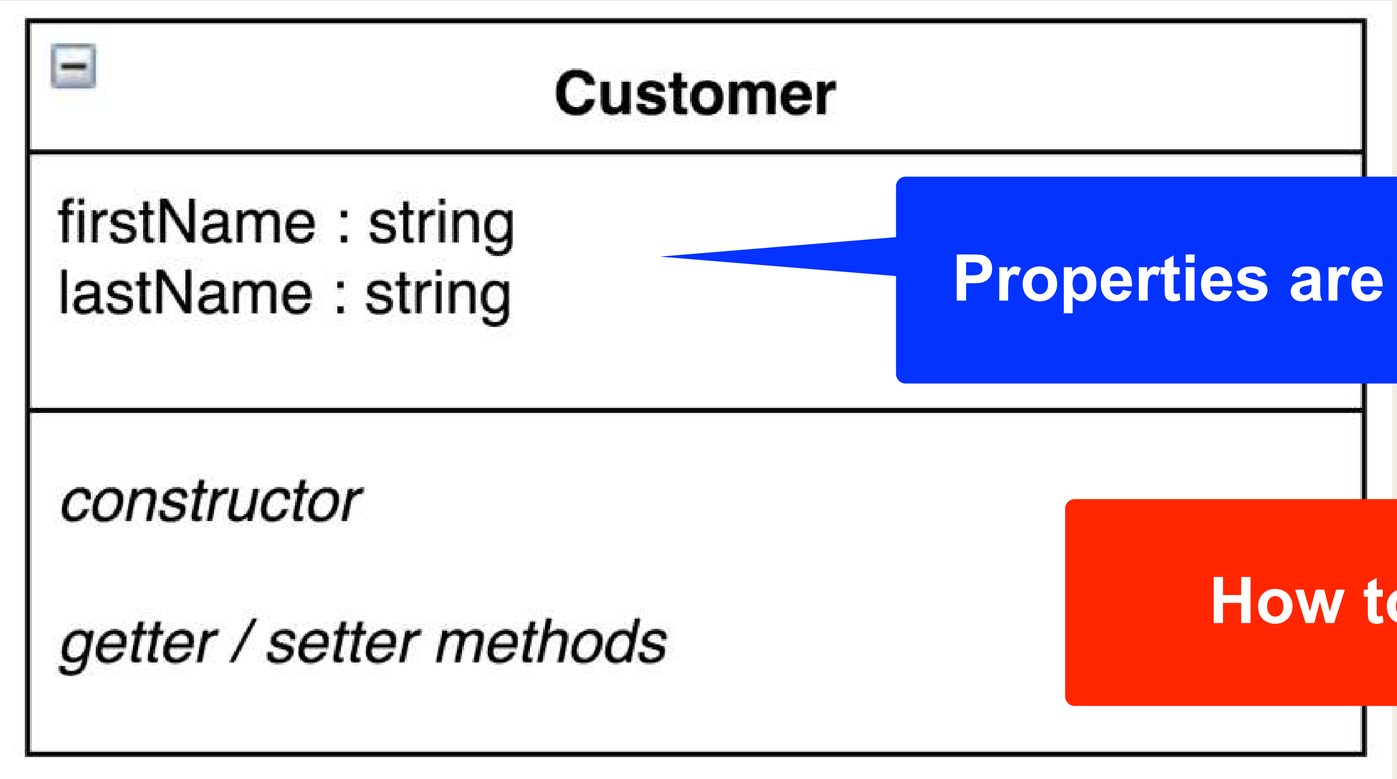
Do not generate .js file
if there is a compilation error

```
C:\> del Customer.js

C:\> tsc --noEmitOnError Customer.ts

Customer.ts:16:12 - error TS2341: Property 'firstName' is private and only
accessible within class 'Customer'.
…

C:\> dir

Customer.ts
```

Yaaay!
The .js file was NOT generated!

TypeScript - Accessors

Customer Class



Getter / Setter Methods

- Since our properties are private, we need a way to access them

- We can create traditional methods as in other OO languages:
  - Define getter/setter methods

Getter / Setter Methods

File: Customer.ts

```
class Custome

    private firstNa e: string;
    private lastNam : string;
    … …

    public getFirstName(): string {
        return this.firstName;
    }
```

Method name

Return type

Getter / Setter Methods

File: Customer.ts

```
class Custome

    private firstNa e: string;
    private lastNam : string;

    public get irstName(): string {
        return this.firstName;
    }

    public setFirstName(theFirst: string): void {
        this.firstName = theFirst;
    }
}
```

Method name

Method name

Return type

Param type

Return type

```
// now let's use it
let myCustomer = new Customer("Martin", "Dixon");

myCustomer.setFirstName("Greg");
console.log(myCustomer.getFirstName());
```

TypeScript: Accessors - Get / Set

- TypeScript also offers an alternate syntax
- Define special: get / set methods
- Known as **Accessors**

TypeScript: Accessors - Get / Set

Can give any internal name

File: Customer.ts

```
class Customer {

    private x: string;
    private y: string;
    … …

    public get firstName(): string {
        return this.x;
    }

    public set firstName(value: string) {
        this.x = value;
    }
}
```

```
// now let's use it
let myCustomer = new Customer("Martin", "Dixon");

myCustomer.firstName = "Susan";
console.log(myCustomer.firstName);
```

The public get/set accessors are still called accordingly

TypeScript: Accessors - Get / Set

Removed “public” on the accessors.
If no access modifier given, “public” by default

File: Cust

```
class Customer {

    private _firstName: string;
    private _lastName: string;
    … …

    get firstName(): string {
        return this._firstName;
    }

    set firstName(value: string) {
        this._firstName = value;
    }
}
```

Renamed to the original names
from previous slides

Compiler flag

- The get/set accessors feature is only supported in ES5 and higher
- You have to set a compiler flag in order to compile the code

```
C:\> tsc --target ES5 --noEmitOnError Customer.ts
```

Compiler flag

Problem with too many compiler flag

- You may have noticed, that we have a lot of compiler flags
  - Too much stuff to remember … easy to forget
- Wouldn't it be great to set this up in a config file?
- TypeScript has a solution: **tsconfig.json** file

tsconfig.json

- **tsconfig.json** file defines compiler options and project settings
- Place this file in the root of your project directory

File: tsconfig.json

```
{
  "compilerOptions": {
    "noEmitOnError": true,
    "target": "es5"
  }
}
```

tsconfig.json

- You can also generate a template for this file

```
C:\> tsc --init
```

Generates a default tsconfig.json file

- Then edit the **tsconfig.json** accordingly for your project requirements

Compiling your Project

- Once your project has a **tsconfig.json** file, then you can compile with

```
C:\> tsc
```

No need to give names of TypeScript files.
By default, will compile all *.ts files

www.luv2code.com/tsconfig-docs

TypeScript: Accessors - Get / Set

File: Customer.ts

```
class Customer {

    private _firstName: string;
    private _lastName: string;
    … …

    get firstName(): string {
        return this._firstName;
    }

    set firstName(value: string) {
        this._firstName = value;
    }
}
```

```
// now let's use it
let myCustomer = new Customer("Martin", "Dixon");

myCustomer.firstName = "Susan";
console.log(myCustomer.firstName);
```

Parameter Properties

Parameter Properties

- TypeScript offers a short-cut syntax for creating constructors
- Helps to minimize the boilerplate code for constructors

Constructor - Traditional Approach

```
class Customer {

    private _firstName: string;
    private _lastName: string;

    constructor(theFirst: string, theLast: string) {
        this._firstName = theFirst;
        this._lastName = theLast;
    }

    // accessors: get/set
    …
}
```

TypeScript offers a shortcut :-)

Constructor - Parameter Properties

Traditional Approach

```
class Customer {

    private _firstName: string;
    private _lastName: string;

    constructor(theFirst: string, theLast: string) {
        this._firstName = theFirst;
        this._lastName = theLast;
    }
      // accessors: get/set
      …
}
```

The Short Cut

```
class Customer {

    constructor(private _firstName: string,
                private _lastName: string) {

    }

    // accessors: get/set
    …
```

Defines properties and
assigns properties automagically.

Minimizes boilerplate coding!

Parameter Properties - In Action

File: Customer.ts

```
class Customer {

    constructor(private _firstName: string,
                private _lastName: string) {

    }

    // accessors: get/set
    …
}
```

Defines properties and assigns properties automagically.

Minimizes boilerplate coding!

Parameter Properties - In Action

File: Customer.ts

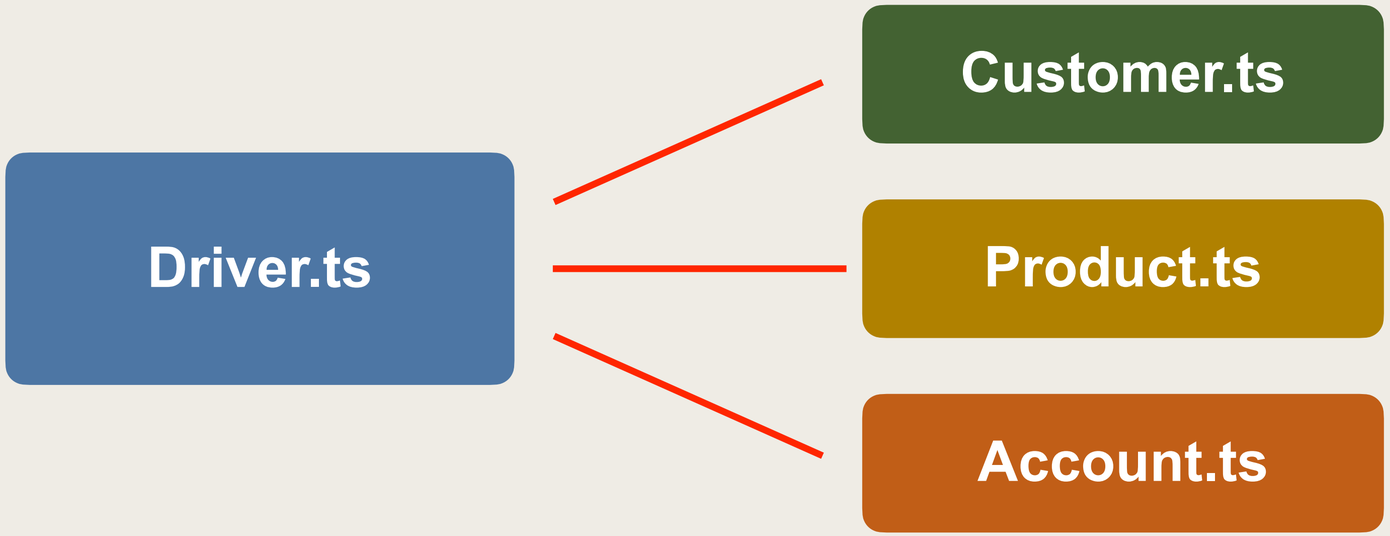
```
class Customer {

    constructor(private _firstName: string,
                private _lastName: string) {

    }

    // accessors: get/set
    …
}
```

```
// now let's use it
let myCustomer = new Customer("Martin", "Dixon");

myCustomer.firstName = "Susan";
console.log(myCustomer.firstName);
```

Modules

Code Organization

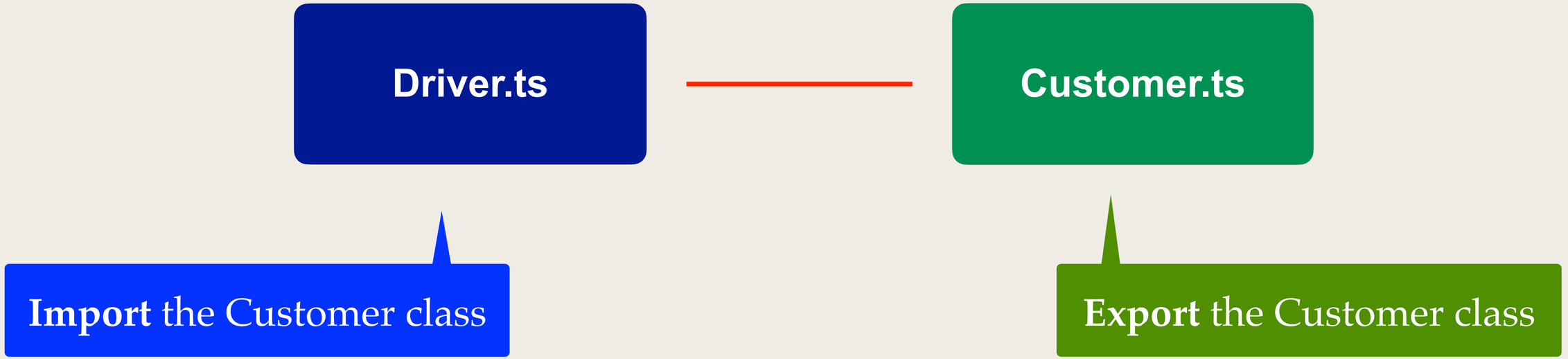
- Currently all of our code is in a single file
- For real-time projects, we would like to place code in separate files



TypeScript Modules

- TypeScript supports the concept of modules
- A module can **export** classes, functions, variables etc
- Another file can **import** classes, functions, variables etc from a module

Example


Driver.ts
Customer.ts
Import the Customer class
Export the Customer class

Module Example

File: Customer.ts

```
export class Customer {

    ...
    ...
    ...

}
```

Export the class

Module Example

Can give
relative directory path

Based on file name
(leave off .ts)

File: Driver.ts

```
import { Customer } from './Customer';

let myCustomer = new Customer("Martin", "Dixon");

console.log(myCustomer.firstName);
console.log(myCustomer.lastName);
```

File: Customer.ts

```
export class Customer {

    ...
    ...
    ...

}
```

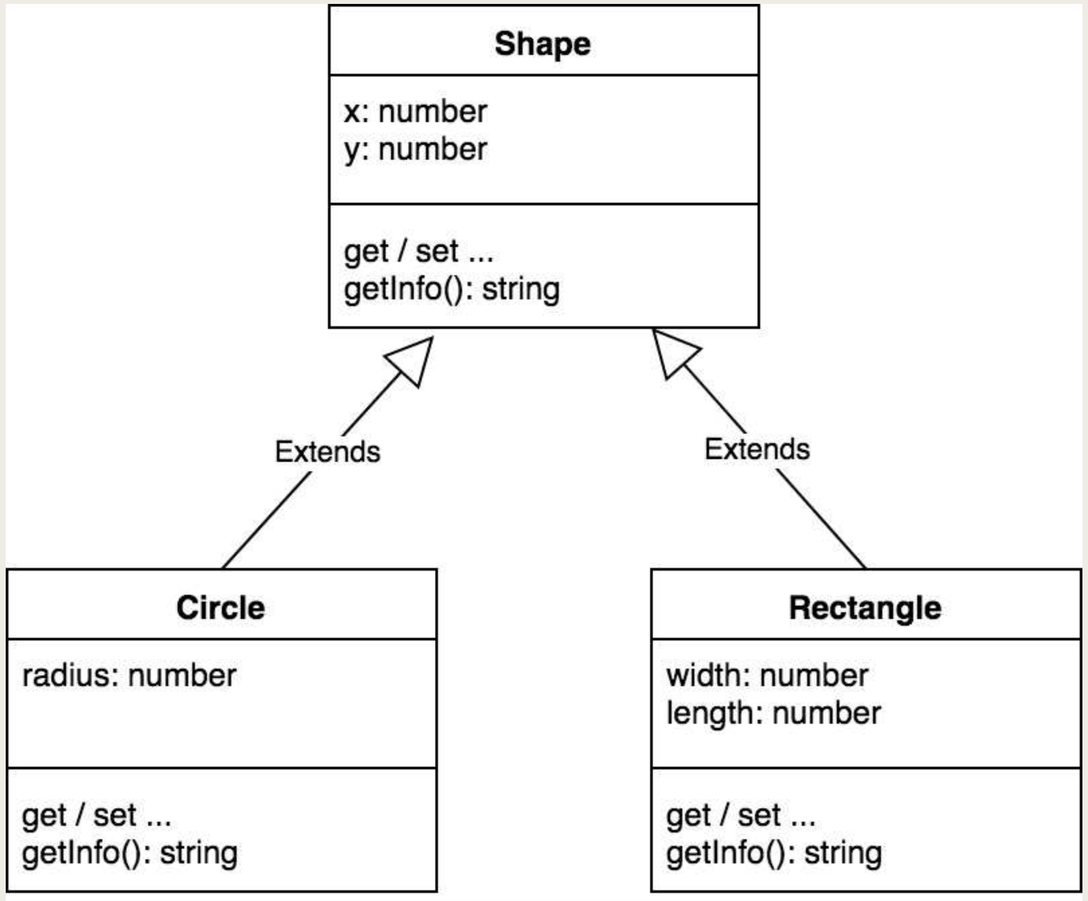
Inheritance

Inheritance

- TypeScript supports the object-oriented concept of inheritance
  - Define common properties and methods in the superclass
  - Subclasses can extend superclasses and add properties and methods
  - Support for abstract classes and overriding

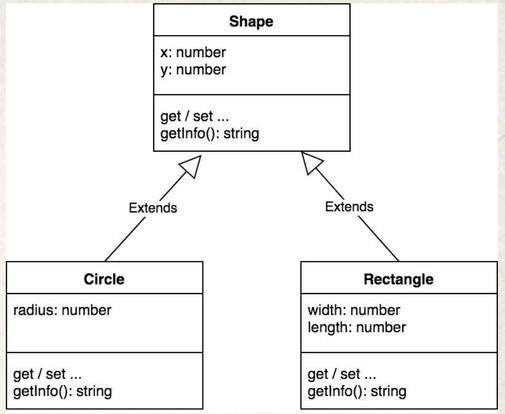
- TypeScript only supports single inheritance
  - However, you can implement multiple interfaces

Inheritance Example



Can override the getInfo() method

Inheritance Example



File: Shape.ts

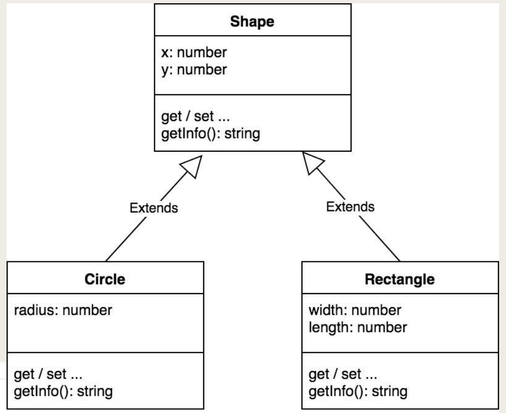
```ts
export class Shape {

    constructor(private _x: number, private _y: number) {
    }

    // get/set accessors ...

    getInfo(): string {
        return `x=${this._x}, y=${this._y}`;
    }
}
```

File: Circle.ts

```ts
import { Shape } from './Shape';

export class Circle extends Shape {

    constructor(theX: number, theY: number,
                private _radius: number) {
        super(theX, theY);
    }

    // get/set accessors ...

    getInfo(): string {
        return super.getInfo() + `, radius=${this._radius}`;
    }
}
```

Override the getInfo() method

Creating a main app



File: Shape.ts

```
export class Shape {
    …
    getInfo() {
        return `x=${this._x}, y=${this._y}`;
    }
}
```

File: Circle.ts

```
import { Shape } from './Shape';

export class Circle extends Shape {
    …
    getInfo() {
        return super.getInfo() + `, radius=${this._radius}`;
    }
}
```

File: Driver.ts

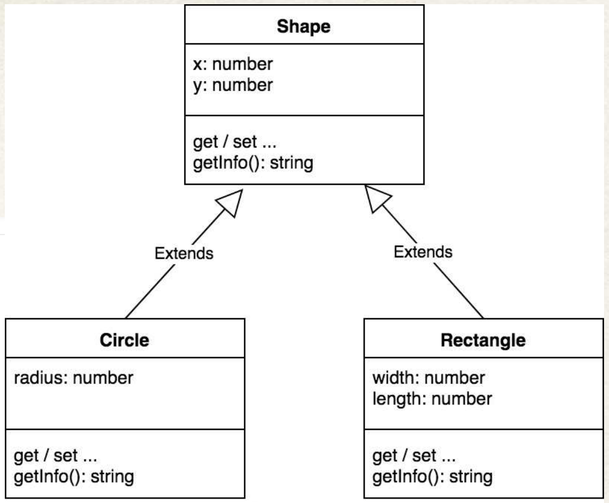
```
import { Shape } from './Shape';
import { Circle } from './Circle';

let myShape = new Shape(10, 15);
console.log(myShape.getInfo());

let myCircle = new Circle(5, 10, 20);
console.log(myCircle.getInfo());
```

```
x=10, y=15
x=5, y=10, radius=20
```

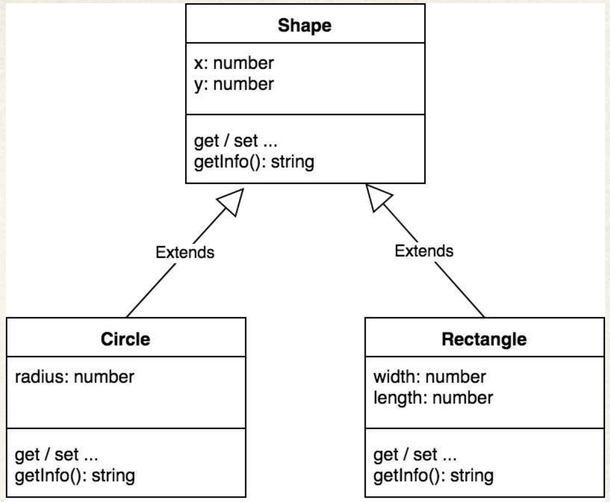
Rectangle

File: Rectangle.ts

```ts
import { Shape } from './Shape';

export class Rectangle extends Shape {

    constructor(theX: number, theY: number,
                private _width: number, private _length: number) {

        super(theX, theY);
    }

    // get/set accessors ...

    getInfo(): string {
        return super.getInfo() + `, width=${this._width}, length=${this._length}`;
    }
}
```



Creating a main app



File: Driver.ts

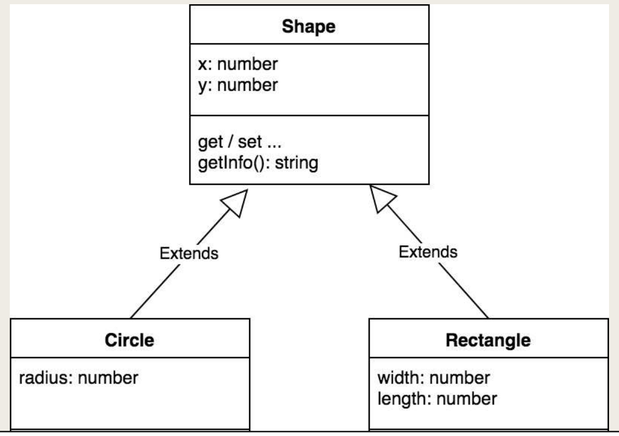
```
import { Shape } from './Shape';
import { Circle } from './Circle';
import { Rectangle } from './Rectangle';

let myShape = new Shape(10, 15);
console.log(myShape.getInfo());

let myCircle = new Circle(5, 10, 20);
console.log(myCircle.getInfo());

let myRectangle = new Rectangle(0, 0, 3, 7);
console.log(myRectangle.getInfo());
```

```
x=10, y=15
x=5, y=10, radius=20
x=0, y=0, width=3, length=7
```

Creating an Array of Shapes

File: ArrayDriver.ts

```
… …

let myShape = new Shape(10, 15);
let myCircle = new Circle(5, 10, 20);
let myRectangle = new Rectangle(0, 0, 3, 7);

// declar… array… shapes … initially empty
let theSha… Sh… = [];

// add the s… to the array
theShapes.…ape);
theShape…h(…le);
theShape…ush(my…tangle);

for (let tempShape of theShapes) {
    console.log(tempShape.getInfo());
}
```




```
// declare an array for shapes
let theShapes: Shape[] = [myShape, myCircle, myRectangle];
```

```
x=10, y=15
x=5, y=10, radius=20
x=0, y=0, width=3, length=7
```

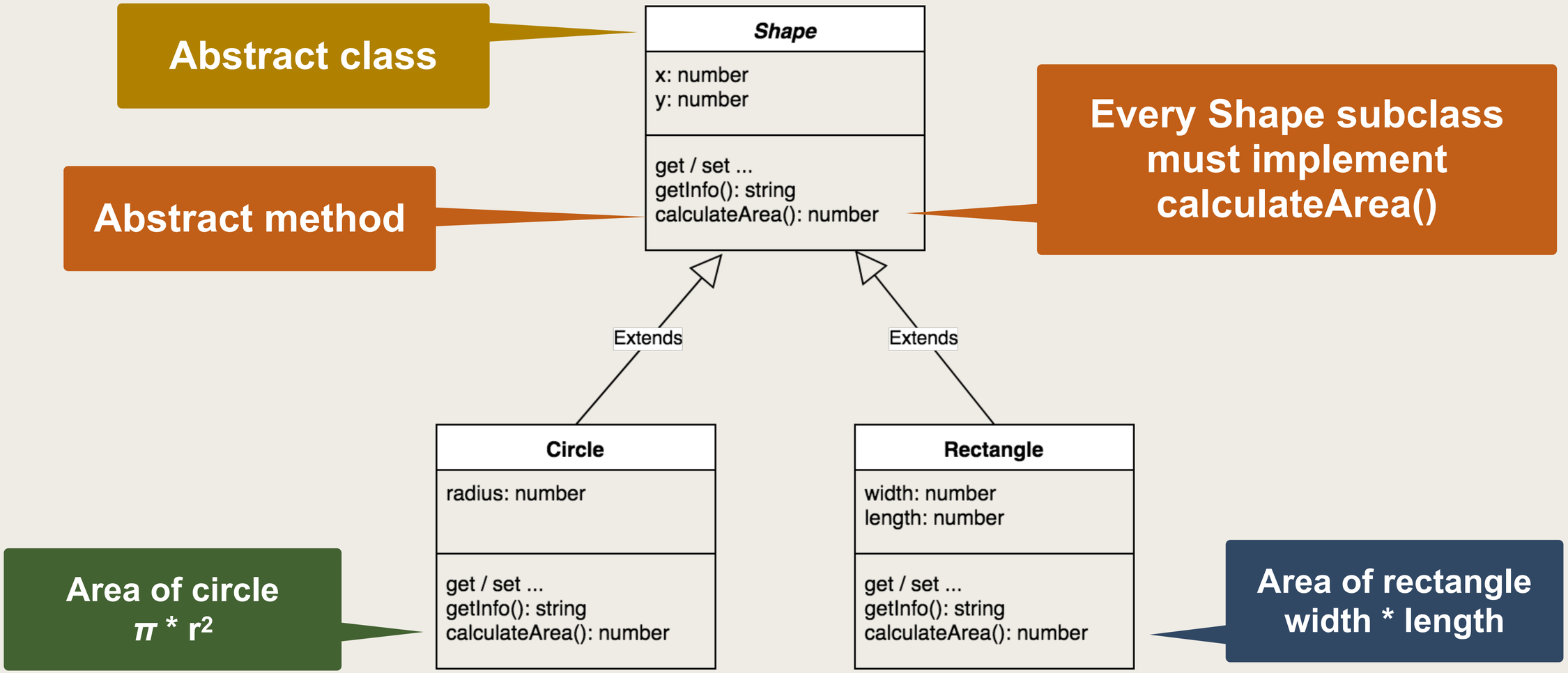
Abstract Classes

Abstract Class

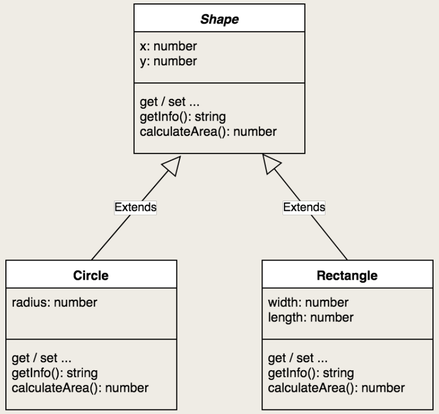
- An abstract class represents a general concept
  - For example: Shape, Vehicle, Computer, etc …

- Can't create an instance of an abstract class
- Abstract class can also have abstract method(s)
- Abstract method must be implemented by concrete subclasses

Abstract Class Example



Abstract Class Example



Mark the class as abstract

File: Shape.ts

```
export abstract class Shape {

    // previous code ...

    abstract calculateArea(): number;
}
```

Abstract method

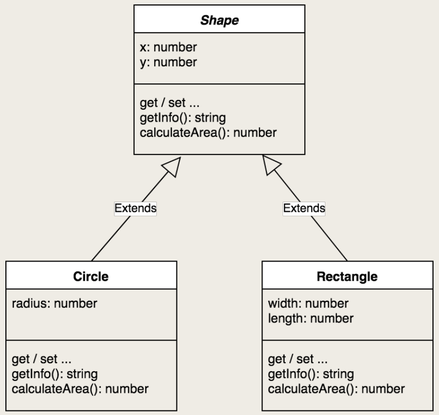
Override the calculateArea() method

File: Rectangle.ts

```
import { Shape } from './Shape';

export class Rectangle extends Shape {

    // previous code ...

    calculateArea(): number {
        return this._width * this._length;
    }
}
```

Circle

`Math` is a built-in object that has properties and methods for mathematical constants and functions



File: Shape.ts

```ts
export abstract class Shape {

    // previous code ...

    abstract calculateArea(): number;
}
```

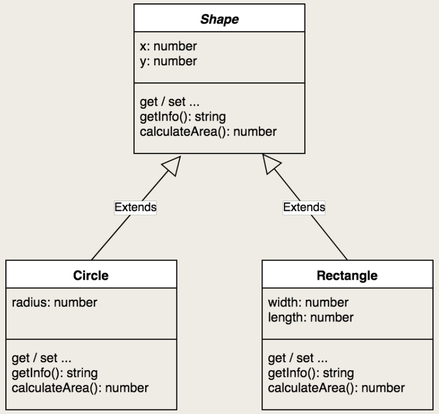
File: Circle.ts

```ts
import { Shape } from './Shape';

export class Circle extends Shape {

    // previous code ...

    calculateArea(): number {
        return Math.PI * Math.pow(this._radius, 2);
    }
}
```

Override the calculateArea() method

Area of circle
$\pi * r^2$

Creating an Instance



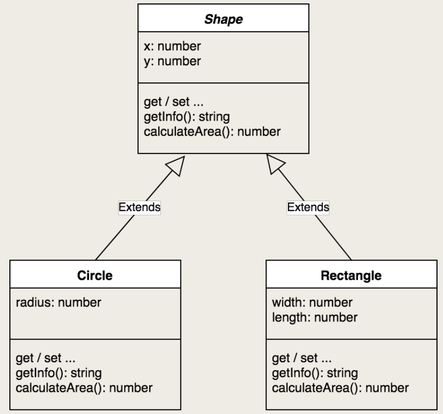

```
let myShape = new Shape(10, 15);
console.log(myShape.getInfo());
```

This will NOT compile since Shape is an abstract class

Can't create instance of abstract class directly

Only concrete subclasses: Circle, Rectangle, …

Creating an Array of Shapes



File: ArrayDriver.ts

```
… …

let myCircle = new Circle(5, 10, 20);
let myRectangle = new Rectangle(0, 0, 3, 7);

// declare an array for shapes … initially empty
let theShapes: Shape[] = [];

// add the shapes to the array
theShapes.push(myCircle);
theShapes.push(myRectangle);
```

```
x=5, y=10, radius=20
Area=1256.6370614359173

x=0, y=0, width=3, length=7
Area=21
```

Area of circle
$\pi * r^2$

Area of rectangle
width * length

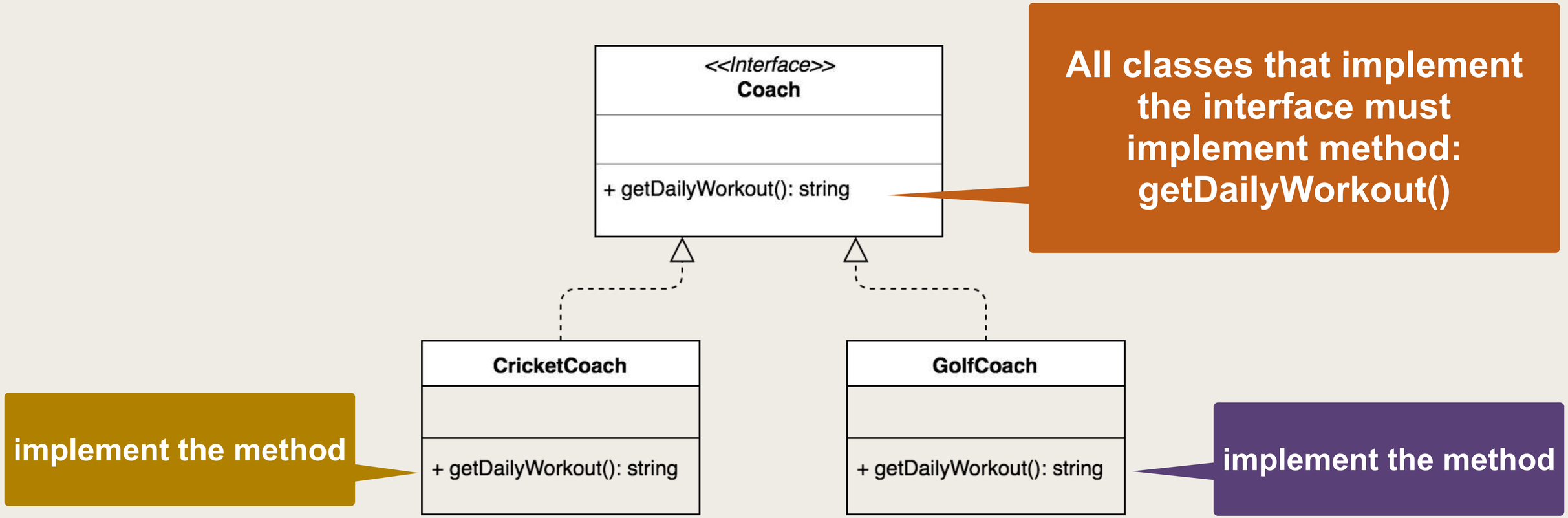
Interfaces

Interfaces

- TypeScript supports interfaces
  - Define an interface with a method contract
  - Classes implement the interface accordingly
  - A class can implement multiple interfaces

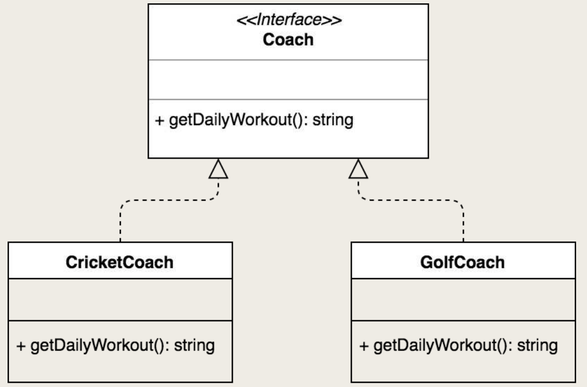
- TypeScript can also use interfaces to support contracts with properties
  - For examples with properties, see:
  - `http://www.typescriptlang.org/docs/handbook/interfaces.html`

Interface Example



Interface Example

Define the interface



File: Coach.ts

```ts
export interface Coach {

    getDailyWorkout(): string;
}
```

Implement the interface

File: CricketCoach.ts

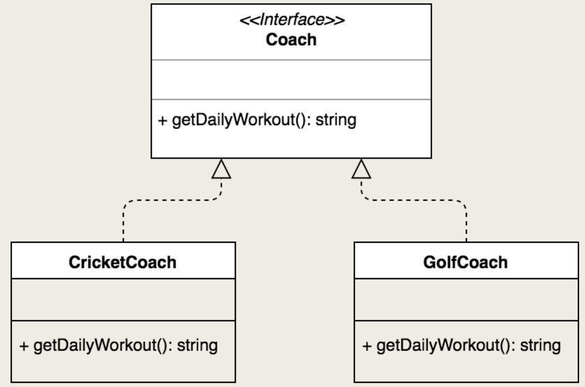
```ts
import { Coach } from "./Coach";

export class CricketCoach implements Coach {

    getDailyWorkout(): string {
        return "Practice your spin bowling technique.";
    }
}
```

Implement the interface

File: GolfCoach.ts

```ts
import { Coach } from "./Coach";

export class GolfCoach implements Coach {

    getDailyWorkout(): string {
        return "Hit 100 balls at the golf range.";
    }
}
```

Creating a Main App

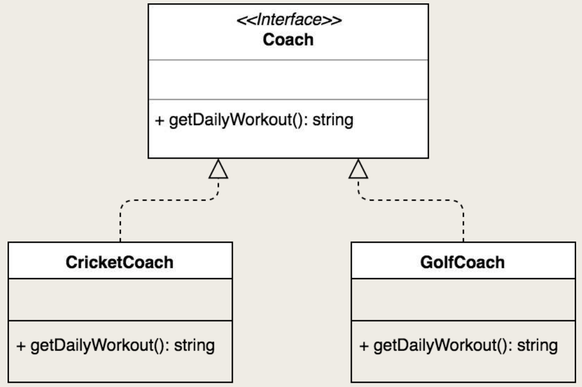


File: Driver.ts

```
import { Coach } from "./Coach";
import { CricketCoach } from "./CricketCoach";
import { GolfCoach } from "./GolfCoach";

let myCricketCoach = new CricketCoach();
console.log(myCricketCoach.getDailyWorkout());

let myGolfCoach = new GolfCoach();
console.log(myGolfCoach.getDailyWorkout());
```

```
Practice your spin bowling technique.
Hit 100 balls at the golf range.
```

Creating an Array of Coaches



File: ArrayDriver.ts

```
… …

let myCricketCoach = new CricketCoach();
let myGolfCoach = new GolfCoach();

// declare an array for coaches ... initially empty
let theCoaches: Coach[] = [];

// add the coaches to the array
theCoaches.push(myCricketCoach);
theCoaches.push(myGolfCoach);

for (let tempCoach of theCoaches) {
    console.log(tempCoach.getDailyWorkout());
}
```

```
Practice your spin bowling technique.
Hit 100 balls at the golf range.
```