TypeScript

Generics

Promises

WebClients

eg43.ts

```
function addNumbers(a:number,b:number):number{
  return a+b;
}
function addAnything(a:any,b:any):any {
  return a+b;
}

console.log(addNumbers(10,20.2));
console.log(addAnything("Good ",20));
```

---

eg44.ts

```
var x=[10,20,30,40];
x.splice(3,1);
console.log(x);
```

---

eg45.ts

```
class Stack<Whatever>{
private x:Whatever[]=[];  // x:Whatever[]; would not make it an array, and length
push(p:Whatever)
{
this.x[this.x.length]=p;
}
pop():Whatever
{
var data:Whatever;
data=this.x[this.x.length-1];
this.x.splice(this.x.length-1,1);
return data;
}
isEmpty():boolean
{
  return this.x.length==0;
}
}
class Student {
  constructor(public rollNumber:number,public name:string){}
}

var k:Stack<number>=new Stack<number>();
k.push(10);
k.push(20);
k.push(30);
while(k.isEmpty()==false)
{
  console.log(k.pop());
}
console.log("-------------------------------");
```

```
var s:Stack<Student>=new Stack<Student>();
s.push(new Student(101,"Sameer"));
s.push(new Student(102,"Rakesh"));
s.push(new Student(103,"Lokesh"));
while(!s.isEmpty())
{
  let t=s.pop();
  console.log(`Roll Number ${t.rollNumber}, Name : ${t.name}`);
}
```

---

eg46.ts

```
interface Vehicle
{
  getDateOfSale():Date;
  getSalePrice():number;
}
function getApproximateValue<T extends Vehicle>(v:T):number {
var d=new Date();
var todayYear=d.getFullYear();
var salePrice=v.getSalePrice();
var ageOfVehicle=todayYear-v.getDateOfSale().getFullYear();
var reducedPrice=salePrice;
for(let i=1;i<=ageOfVehicle;i++)
{
  reducedPrice=reducedPrice-(reducedPrice*.05); // 2 % per year
}
return Math.floor(reducedPrice);
}

class CBZ implements Vehicle{
  constructor(private price:number,private dateOfSale:Date){}
  getSalePrice():number
  {
    return this.price;;
  }
  getDateOfSale():Date{
    return this.dateOfSale;
  }
  // some more methods according to CBZ
}

class HondaCity implements Vehicle{
  constructor(private price:number,private dateOfSale:Date){}
  getSalePrice():number
  {
    return this.price;;
  }
```

```
  getDateOfSale():Date{
    return this.dateOfSale;
  }
  // some more methods according to HondaCity
}

var h=new HondaCity(1200000,new Date(2012,0,1)); // y,m (zero based),d
var m=new CBZ(90000,new Date(2016,0,1));
console.log("Today the approximate price of this HondayCity is "+getApproximateValue(h));
console.log("Today the approximate price of this HondayCity is "+getApproximateValue(m));
```

eg47.ts

```
function linearSort(x:number[]) {
for(let e=0;e<=x.length-2;e++){
for(let f=e+1;f<=x.length-1;f++)
if(x[f]<x[e]) [x[e],x[f]]=[x[f],x[e]];
}
}

var a:number[]=[10,3,40,20,30,50,32,23,53,21];
linearSort(a);
console.log(a);
```

eg48.ts

```
interface Comparator<T>
{
  compare(leftOperator:T,rightOperand:T):number;
}
function linearSort<T>(x:T[],fn:Comparator<T>) {
for(let e=0;e<=x.length-2;e++){
for(let f=e+1;f<=x.length-1;f++)
if(fn.compare(x[f],x[e])<0) [x[e],x[f]]=[x[f],x[e]];
}
}
class NumberComparator implements Comparator<number>
{
  compare(leftOperand:number,rightOperand:number):number {
    return leftOperand-rightOperand;
  }
}
class StringComparator implements Comparator<string>
{
  compare(leftOperand:string,rightOperand:string):number {
    return leftOperand.localeCompare(rightOperand);
  }
}
class Student{
  constructor(private rollNumber:number,private name:string){}
```

```
  getRollNumber(){ return this.rollNumber; }
  getName() { return this.name};
}
class StudentComparator implements Comparator<Student>
{
  compare(leftOperand:Student,rightOperand:Student):number {
    return leftOperand.getName().localeCompare(rightOperand.getName());
  }
}
var a:number[]=[10,3,40,20,30,50,32,23,53,21];
linearSort(a,new NumberComparator);
console.log(a);
var b:string[]=["Sourabh","Amit","Rupesh","Salim","Avinash","Ram","Shyam"];
linearSort(b,new StringComparator);
console.log(b);
var s:Student[]=[new Student(101,"Sameer"),new Student(102,"Amit"),new Student(103,"Bobby")];
linearSort(s,new StudentComparator);
for(let st of s)
{
  console.log(`Roll number : ${st.getRollNumber()}, Name : ${st.getName()}`);
}
```

Assignment :

1) **Implement Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Radix Sort, Merge Sort and Heap Sort**
2) **Implement SinglyLinked List, Doubly Linked List, BinarySearchTree (AVL)**

eg49.ts

```
class Bulb
{
constructor(public wattage:number,public brand:string,public price:number){}
}

var b1=new Bulb(100,"Philips",50);
var b2=new Bulb(60,"Sylvania",35);
console.log("Wattage : "+b1["wattage"]+", Brand : "+b1["brand"]+",Price : "+b1["price"]);
console.log("Wattage : "+b2["wattage"]+", Brand : "+b2["brand"]+",Price : "+b2["price"]);
```

eg50.ts

```
function getProperty<T,P extends keyof T>(instance:T,property:P)
{
  return instance[property];
}
class Bulb
{
constructor(public wattage:number,public brand:string,public price:number){}
}
class Vehicle
{
constructor(public price:number,public brand:string,public type:string){}
}
var b1=new Bulb(100,"Philips",50);
var b2=new Bulb(60,"Sylvania",35);
var v1=new Vehicle(90000,"CBZ","MotorCycle");
var v2=new Vehicle(120000,"Honda City","Car");
console.log("Brand : "+getProperty(b1,"brand")+", Price : "+getProperty(b1,"price"));
console.log("Brand : "+getProperty(b2,"brand")+", Price : "+getProperty(b2,"price"));
console.log("Brand : "+getProperty(v1,"brand")+", Price : "+getProperty(v1,"price")+",
Type :"+getProperty(v1,"type"));
console.log("Brand : "+getProperty(v2,"brand")+", Price : "+getProperty(v2,"price"));
// following line won't compile, as age is not defined as property of vehicle
// console.log("Age : "+getProperty(v2,"age"));
```

Now whatever we will be doing will require DOM, XMLHttpObject etc. which is available in Browser. We can do the same without browser, but going on that track is not worth it.

I will be using Tomcat8 as server as near to zero examples are available (from zero up)
We will write the client side with nothing else but HTML5 & TypesSript.

We need to stick to learning TypeScript, so instead of writing servlets for serverside, I will just hookup dirty coding in JSP files. (Note : I do not promote such coding style, it should have been servlets or webservices etc, but I don't want to waste time in it right now.

I have configured tomcat8 on c:\

in c:\tomcat8\webapps create folder named as tseg with a folder WEB-INF in it. In tseg folder create a folder named as js.

Move into the js folder (on command prompt/terminal) and type

tsc --init

a tsconfig.json file will be created, open it and against the compileoptions, change the value of target from es05 to ES2017.
Now start ATOM, remove our tseg folder from it, by right clicking the folder in explorer and selecting

remove project folder. Then select File, followed by Open Folder and select c:\tomcat8\webapps\tseg\js folder (or whatever is your path in UNIX/LINUX)

Now create the following eg51.ts files

eg51.ts

```
class Student
{
  constructor(public rollNumber:number,public name:string,public city:string){}
}
function getData(url:string):Promise<any> {
  let assurance:Promise<any>;
  assurance=new Promise(function(resolve,reject){
  let request=new XMLHttpRequest();
  // setting up the onload event
  request.onload=function(){
    if(request.status==200)
    {
      resolve(request.response);
    }else{
    reject(new Error(request.statusText));
    }
  };
  // setting up the onerror event
  request.onerror=function(){
    reject(new Error("Request error "+request.statusText));
  }
  request.open("GET",url);
  request.send();
  });
  return assurance;
}

function eg51(){
let promise=getData("eg51.jsp");
promise.then(function(response){
let students:Student[];
students=JSON.parse(response);
for(let student of students)
{
  alert(`Roll number : ${student.rollNumber}, Name : ${student.name}, City : ${student.city}`);
}
}).catch(function(error){
alert(error);
});
}
```

To transpile type (while staying in the tseg\js folder)

tsc --target ES2017

The eg51.js shoud get generated. Now in tseg folder of webapps create the following files.

eg51.jsp

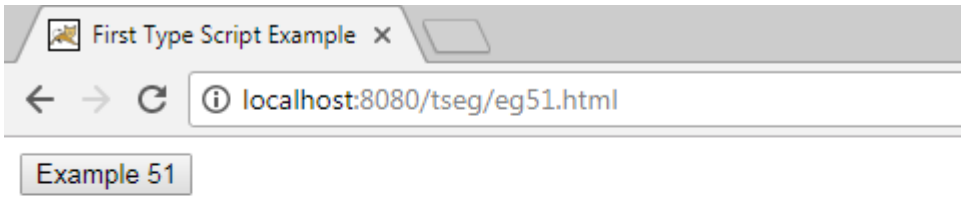
```
<%
try
{
Thread.sleep(3000); // just creating a time gap of 3 seconds, before the response is sent
}catch(InterruptedException ie)
{
}
%>
[{
"rollNumber": 101,
"name": "Sameer",
"city": "Ujjain"
},
{
"rollNumber": 102,
"name": "Gopal",
"city": "Ujjain"
},
{
"rollNumber": 103,
"name": "Lokesh",
"city": "Indore"
},
{
"rollNumber": 104,
"name": "Rahul",
"city": "Ujjain"
}]
```

---

index.html

```
<!doctype html>
<html lang="en">
<head>
<meta charset='utf-8'>
<title>Type Script Examples</title>
<script src='js/eg51.js'></script>
</head>
<body>
<button type='button' onclick='eg51()'>Example 51</button>
</body>
</html>
```

Start tomcat,
type the URL in address bar of Google Chrome
http://localhost:8080/tseg/eg51.html
You should see a button as follows



Click the button, be patient, we have added a sleep of 3 seconds on the server side, after that time, you should get to see the following (a messagebox should appear 4 times, everytime displaying the data of a student.

Thats it, now everything is working fine, the transpiled JS is working for us.

Create the following file in tseg\js folder

eg52.ts

```
class ResponseWrapper
{
  constructor(public successful:boolean,public text:string){}
}
function objectToPostArguments(object:any)
{
return Object.keys(object).map(function(key) {
return encodeURIComponent(key) + '=' +
encodeURIComponent(object[key]);
}).join('&');
}

function postData(url:string,student:Student):Promise<any> {
  let assurance:Promise<any>;
  assurance=new Promise(function(resolve,reject){
  let request=new XMLHttpRequest();
  // setting up the onload event
  request.onload=function(){
    if(request.status==200)
    {
      resolve(request.response);
    }else{
```

```
   reject(new Error(request.statusText));
   }
  };
  // setting up the onerror event
  request.onerror=function(){
   reject(new Error("Request error "+request.statusText));
  }
  request.open("POST",url);
  request.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  request.send(objectToPostArguments(student));
  });
  return assurance;
}

function eg52(){
let
rollNumberComponent:HTMLInputElement=<HTMLInputElement>document.getElementById("rollN
umber");
let rollNumber=parseInt(rollNumberComponent.value);

let nameComponent:HTMLInputElement=<HTMLInputElement>document.getElementById("name");
let name=nameComponent.value;

let cityComponent:HTMLInputElement=<HTMLInputElement>document.getElementById("city");
let city=cityComponent.value;

let student=new Student(rollNumber,name,city);
let promise=postData("eg52.jsp",student);
promise.then(function(response){
let responseWrapper:ResponseWrapper=JSON.parse(response);
if(responseWrapper.successful){
  alert(responseWrapper.text);
} else{
  alert(`Problem : ${responseWrapper.text}`);
}
}).catch(function(error){
alert(error);
});
}
```

Note : we are using Student class, which has been defined in eg51.ts, so to transpile type

```
tsc  --target  ES2017   eg51.ts   eg52.ts
```

Create the following JSP file in tseg folder

eg52.jsp

```
<%
int rollNumber=Integer.parseInt(request.getParameter("rollNumber"));
String name=request.getParameter("name");
String city=request.getParameter("city");
System.out.println("Name : "+name);
System.out.println("Roll number : "+rollNumber);
System.out.println("City : "+city);
if(rollNumber>=101 && rollNumber<=104)
{
%>
{ "successful": false,"text":"Roll Number, exists" }
<%
}
else
{
%>
{ "successful": true,"text":"Student added" }
<%
}
%>
```

Now include the transpiled eg52.js in your index.html using the following just below the line where we had included eg51.js (as follows)

```
<script src='js/eg51.js'></script>   <---- this line already exists in the index.html file
<script src='js/eg52.js'></script>
```

Now just below the button tag in index.html add the following

```
<hr>
<h1>Example 52</h1>
Roll Number <input type='number' id='rollNumber'><br>
Name <input type='text' id='name'><br>
City <input type='text' id='city'><br>
<button type='button' onclick='eg52()'>Add student</button>
<hr>
```

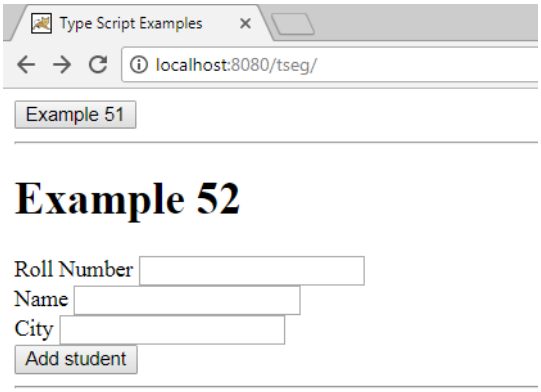
Finally the index.html looks as follows.
(Note from next time, I won't be putting the whole index.html over here)

index.html.

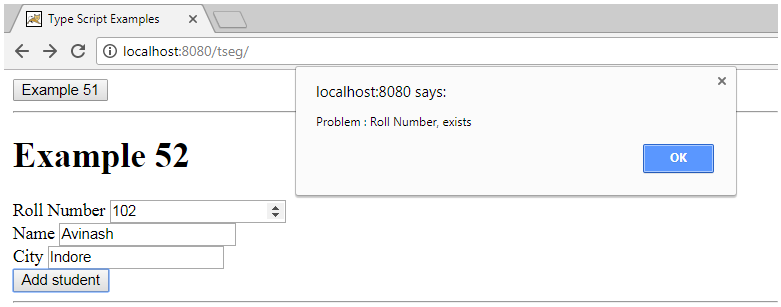
```
<!doctype html>
<html lang="en">
```

```
<head>
<meta charset='utf-8'>
<title>Type Script Examples</title>
<script src='js/eg51.js'></script>
<script src='js/eg52.js'></script>
</head>
<body>
<button type='button' onclick='eg51()'>Example 51</button>
<hr>
<h1>Example 52</h1>
Roll Number <input type='number' id='rollNumber'><br>
Name <input type='text' id='name'><br>
City <input type='text' id='city'><br>
<button type='button' onclick='eg52()'>Add student</button>
<hr>
</body>
</html>
```

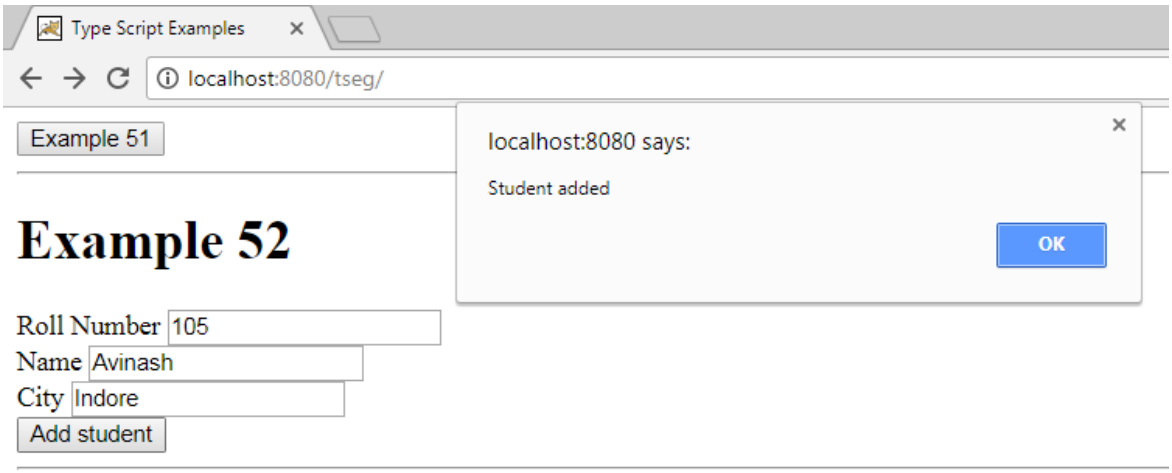
Now start server, and visit tseg and you should see the following



Type the student data and click the Add Student button
Note : We have written the jsp in such a way that if 101,102,103 or 104 is the given roll number then following will be the output.



And if you provide roll number other than (101,102,103 and 104) then following will be the output



Create the following in tseg\js folder

eg53.ts

```
function postJSON(url:string,student:Student):Promise<any> {
  let assurance:Promise<any>;
  assurance=new Promise(function(resolve,reject){
  let request=new XMLHttpRequest();
  // setting up the onload event
  request.onload=function(){
    if(request.status==200)
    {
      resolve(request.response);
    }else{
   reject(new Error(request.statusText));
    }
  };
  // setting up the onerror event
  request.onerror=function(){
    reject(new Error("Request error "+request.statusText));
  }
  request.open("POST",url);
  request.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  request.send(JSON.stringify(student));
  });
  return assurance;
}
function eg53(){
let
rollNumberComponent:HTMLInputElement=<HTMLInputElement>document.getElementById("rollN
```

```
umber");
let rollNumber=parseInt(rollNumberComponent.value);

let nameComponent:HTMLInputElement=<HTMLInputElement>document.getElementById("name");
let name=nameComponent.value;

let cityComponent:HTMLInputElement=<HTMLInputElement>document.getElementById("city");
let city=cityComponent.value;

let student=new Student(rollNumber,name,city);
let promise=postJSON("eg53.jsp",student);
promise.then(function(response){
let responseWrapper:ResponseWrapper=JSON.parse(response);
if(responseWrapper.successful){
  alert(responseWrapper.text);
} else{
  alert(`Problem : ${responseWrapper.text}`);
}
}).catch(function(error){
alert(error);
});
}
```

We are using Student from eg51.ts and ResponseWrapper from eg52.ts, hence to transpile type

tsc --target ES2017 eg51.ts eg52.ts eg53.ts

Create the following jsp file in tseg folder

eg53.jsp

```
<%@page import="java.io.*" %>
<%
StringBuffer sb=new StringBuffer();
BufferedReader br=request.getReader();
String line=null;
while(true)
{
line=br.readLine();
if(line==null) break;
sb.append(line);
}
String rawData=sb.toString();
System.out.println(rawData);
/*
Now you can use the SimpleJSON or Jackson parsers to
parse the JSON to an object and do whatever you want to do with
it. This is not our concern right now.
Right now I will just send the success response.
*/
```

%>
{ "successful":true, "text": "Student added" }

---

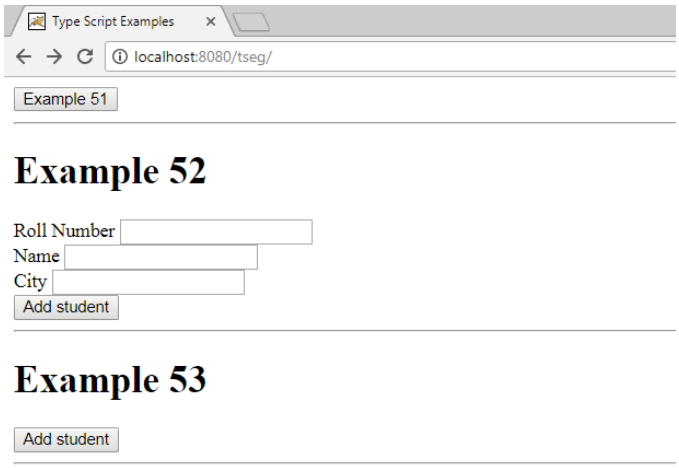
add the line in **index.html** to include eg53.js just below the line on which we have included eg52.js

<script src='js/eg53.js'></script>

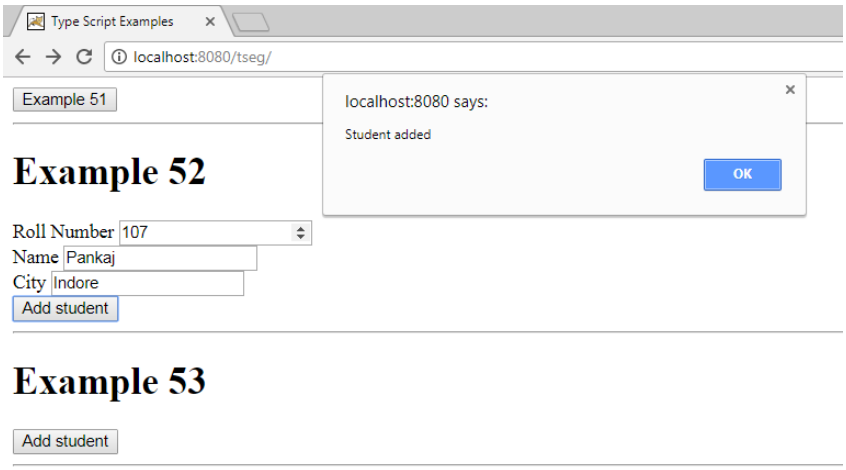
Below the <hr> which follows the button for example eg52, add the following

<h1>Example 53</h1>
<button type='button' onclick='eg53()'>Add student</button>
<hr>

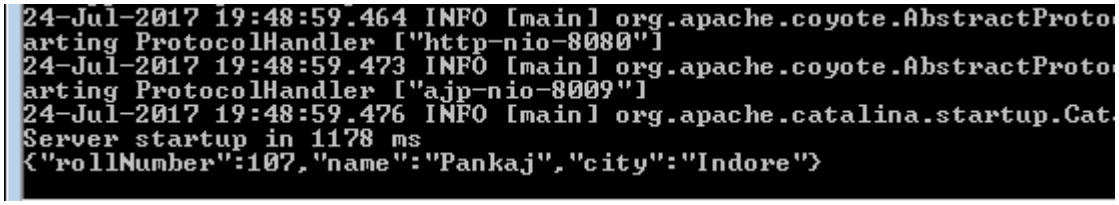
Now start server and visit tseg and you should see the following



Now type the data, against the form in Example 52, but click the button agains Example 53, and you should see the following



Now see the server window, if not visible (for linux/unix or installed as a window server), then open up the necessary log files where tomcat or whatever is you J2EE compliant webserver writes the console output and you should see the following in it



Now it is time to go for Angular 2
