# Exercises: Decorators

## Logging Decorator

Create a simple class **Person** with:

* **public fName: string** - first name
* **public lName: string** - last name
* **static getFullname: string** – takes 2 names and returns the fullname

Create a decorator **@log** and add it to **getFullName** method, when the method is called you should print a message to the console with the following format **"Function '<method name>' called with arguments: <arguments>"** where **arguments** are the arguments the method was called with joined by **", "**

### Examples

|  |
| --- |
| **Input** |
| let person = new Person('John', 'Does');  Person.getFullName(person.fName, person.lName)  Person.getFullName('Benny', 'Tres'); |
| **Output** |
| Function 'getFullName' called with arguments: John, Does  Function 'getFullName' called with arguments: Benny, Tres |

## Validate Age

You are given the class Age, write a decorator on the age setter and add validation checks:

* If the new age value is **less than 1 or more than 200**, **throw an error**, otherwise set the new age value

|  |
| --- |
| **Age** |
| class Age {      private \_age!: number;      constructor(age: number){          this.age = age;       }      set age(val: number){ this.\_age = val; }      get age() { return this.\_age;  }  } |

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| let ageVal = new Age(10);  ageVal.age = -10; | Error: Age must be between 1 and 200 |

## Created On

Implement a simple class **User** with:

* **name: string**
* **age: number**
* **displayUserInfo()** – logs info about the user’s name and age on the console

Add a class decorator that adds a new property to the instance when creating a new user:

* **createdOn**: Date – the current date when creating the instance

**Note:** You’ll need to use type assertions to use the property, asTS will not recognize it, since it’s not in the class declaration.

### Examples

|  |
| --- |
| **Input** |
| const user1 = new User("John Doe", 30);  user1.displayUserInfo()  console.log(user1);  console.log((user1 as any).createdOn); |
| **Output** |
| John Doe, Age: 30  User {name: 'John Doe', age: 30, createdOn: Thu May 01 2025 13:22:52 GMT+0300 (Eastern European Summer Time)}  Thu May 01 2025 13:22:52 GMT+0300 (Eastern European Summer Time) |

## Authorization

You are given a **MockAuthorizationService**:

|  |
| --- |
| **MockAuthrizationService** |
| class MockAuthrizationService {      constructor(private userRole: 'Guest' | 'PersonalDataAdministrator' | 'Admin') { }      canViewData(property: string) {          switch (this.userRole) {              case 'Admin': return true;              case 'PersonalDataAdministrator': return ['name', 'age'].includes(property);              default: return false;          }      }  } |

Write a class **User** with private properties **name**, **age** and **creditCardNumber** and 3 public getters:

* **get name: string**
* **get age: number**
* **get creditCardNumber: string**

Create a decorator factory and add it to all the getters. The decorator factory should take in an instance of **MockAuthorizationService** and return a decorator that uses the **MockAuthorizationService** instance to determine if the property value should be returned or not:

* in case a property is **allowed** to be read, **return the normal value**
* in case a property is **not allowed** to be read, **throw** an error with message **'You are not authorized to view this information'**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| let mockAuthorizationService = new MockAuthrizationService('Admin');  …  const user1 = new User("John Doe", 30, 'ABCD-1234');  console.log(user1.name);  console.log(user1.age);  console.log(user1.creditCardNumber); | John Doe  30  ABCD-1234 |
| let mockAuthorizationService = new MockAuthrizationService('PersonalDataAdministrator');  …  const user1 = new User("John Doe", 30, 'ABCD-1234');  console.log(user1.name);  console.log(user1.age);  console.log(user1.creditCardNumber); | John Doe  30  // Runtime Error: 'You are not authorized to view this information' |
| let mockAuthorizationService = new MockAuthrizationService('Guest');  …  const user1 = new User("John Doe", 30, 'ABCD-1234');  console.log(user1.name);  console.log(user1.age);  console.log(user1.creditCardNumber); | // Runtime Error: 'You are not authorized to view this information' |

## Weather Data

You are given a simple **MockWeatherDataService**:

|  |
| --- |
| **MockWeatherDataService** |
| class MockWeatherDataService {      private weatherData: string[] = [          'Sunny 8° to 20°',          'Partially Cloudy 7° to 19°',          'Sunny 5° to 18°'      ];      addWeatherData(data: string){ this.weatherData.push(data);  }      getWeatherData() { return this.weatherData; }  } |

Create a decorator/decorator factory for the **getWeatherData** method, that caches the data and also stores information when the real data was last served using the following logic:

* If the method is called and there is no cached data, call the **getWeatherData** method get the data, cache a copy of it and mark down the time, then return the real data.
* If the method is called and there is cached data, compare the current time to the time the data was cached:
  + If the data is newer than 5 seconds, print **'Returned from cache'**, then return the cached copy and do not call **getWeatherData**
  + If the data is older than 5 seconds, call the **getWeatherData** method get the newer version of the data, cache a copy of it and mark down the time, then return the real data.

**Note**: You can check [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/setSeconds#description) Date method to help you for the Date comparisons.

### Examples

|  |
| --- |
| **Input** |
| let service = new MockWeatherDataService();  console.log(service.getWeatherData())  console.log(service.getWeatherData())  service.addWeatherData('Partially Cloudy 5° to 11°');  console.log(service.getWeatherData())  //7 seconds later  setTimeout(() => console.log(service.getWeatherData()), 7000) |
| **Output** |
| ['Sunny 8° to 20°', 'Partially Cloudy 7° to 19°', 'Sunny 5° to 18°']  Returned from cache  ['Sunny 8° to 20°', 'Partially Cloudy 7° to 19°', 'Sunny 5° to 18°']  Returned from cache  ['Sunny 8° to 20°', 'Partially Cloudy 7° to 19°', 'Sunny 5° to 18°']  //7 seconds later  ['Sunny 8° to 20°', 'Partially Cloudy 7° to 19°', 'Sunny 5° to 18°', 'Partially Cloudy 5° to 11°'] |

## Flexible Validation

You are given the simple class **User**:

|  |
| --- |
| **User** |
| class User {      private \_name!: string;      private \_age!: number;      private \_password!: string;      constructor(name: string, age: number, password: string) {          this.name = name;          this.age = age;          this.password = password;      }      set name(val: string) { this.\_name = val; }      set age(val: number) { this.\_age = val; }      set password(val: string) { this.\_password = val; }      get name() { return this.\_name; }      get age() { return this.\_age; }  } |

Create decorator factories to validate the **setters** of the **User** class, using flexible requirements:

* **set name –** should validate that the length of the name is at least **minLength** characters long, where **minLength** is a parameter of the decorator factory
* **set age –** should validate that the age is between **[min…max]** where **min** and **max** are parameters of the decorator factory
* **set password –** should validate that the password matcheda given **regex** where the **regex** is a parameter of the decorator factory

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| // minLength = 1  // min = 1, max = 150  // regex = /^[a-zA-Z0-9!@]+$/g  let user = new User('John', 130, 'hardPassword12');  let user2 = new User('John', 30, '!test');  let user3 = new User('John', 25, '@werty');  let user4 = new User('Jo', 20, 'password123'); |  |
| // minLength = 3  // min = 1, max = 100  // regex = /^[a-zA-Z0-9]+$/g  let user = new User('John', 130, 'hardPassword12');  let user2 = new User('John', 30, '!test');  let user3 = new User('John', 25, '@werty');  let user4 = new User('Jo', 20, 'password123'); | //Runtime Error: age must be between 1 and 100  //Runtime Error: password needs to match /^[a-zA-Z0-9]+$/g  //Runtime Error: password needs to match /^[a-zA-Z0-9]+$/g  //Runtime Error: name must have a min length of 3 characters |

## Censored Data

You are given a **MockCensorService** and 3 **data** **classes**:

|  |
| --- |
| **MockCensorService** |
| class **MockCensorService**<T extends { [key: string]: any }> {      constructor(private censoredProperties: string[]) { }      censorProperties(items: T[]) {          let censoredItems = items.slice();          censoredItems.forEach(i => {              this.censoredProperties.forEach(prop => { delete i[prop]; });          });          return censoredItems;      }  } |

|  |
| --- |
| **Code** |
| class **User** {      constructor(  public name: string, public age: number, public creditCardNumber: string) { }    getInfo() { return `${this.name}, Age: ${this.age} CreditCardNumber: ${this.creditCardNumber}`; }  }  class **Employee** {      constructor(public name: string, public birthday: Date, public salary: number) { }        getInfo() { return `${this.name}, Birthday: ${this.birthday?.toLocaleDateString()} Salary: ${this.salary}`; }  }  class **UsersService** {      private \_users: User[];      private \_employees: Employee[];      constructor(users: User[], employees: Employee[]) {  this.\_users = users; this.\_employees = employees;  }      addUser(user: User) { this.\_users.push(user); }      addEmployee(employee: Employee) { this.\_employees.push(employee); }      getUsers() { return this.\_users; }      getEmployees() { return this.\_employees; }  } |

Create decorators/decorator factories on the data classes and methods to achieve the following functionality:

* **getUsers** – should return all users **created in the last 5 seconds** with **some of their fields censored** (which fields to censor is specified by the MockCensoreService instance, check the Examples)
* **getEmployees** – should **log a message** that the method was called and return all employees **created in the last 10 seconds** with **some of their fields censored** (which fields to censor is specified by the MockCensoreService instance, check the Examples)

### Examples

|  |
| --- |
| **Input** |
| let userCensorService = new MockCensorService<**User**>([**'creditCardNumber'**])  let employeeCensorService = new MockCensorService<**Employee**>([**'birthday'**, **'salary'**])  …  const user1 = new User('John Does', 30, 'ABCD-1234');  const user2 = new User('Benny Tres', 23, 'EFGH-5678');  const emp1 = new Employee('Sarah Connor', new Date(1964, 4, 15), 2500);  const emp2 = new Employee('Arnold Schwarzenegger', new Date(1947, 6, 30), 3500);  let usersService = new UsersService([user1, user2], [emp1, emp2]);  let users = usersService.getUsers();  console.log(users.map(x => x.getInfo()));  let employees = usersService.getEmployees();  console.log(employees.map(x => x.getInfo()));  //7 seconds later  setTimeout(() => {      const user3 = new User('Jimmy Quatro', 27, 'IJKL-9012');      const emp3 = new Employee('Kyle Reese', new Date(2004, 0, 1), 2000);      usersService.addUser(user3);      usersService.addEmployee(emp3);      let users = usersService.getUsers();      console.log(users.map(x => x.getInfo()));      let employees = usersService.getEmployees();      console.log(employees.map(x => x.getInfo()));  }, 7000)  //15 seconds later  setTimeout(() => {      let users = usersService.getUsers();      console.log(users.map(x => x.getInfo()));      let employees = usersService.getEmployees();      console.log(employees.map(x => x.getInfo()));  }, 15000) |
| **Output** |
| (2) ['John Does, Age: 30 CreditCardNumber: undefined', 'Benny Tres, Age: 23 CreditCardNumber: undefined']  Method getEmployees called successfully  (2) ['Sarah Connor, Birthday: undefined Salary: undefined', 'Arnold Schwarzenegger, Birthday: undefined Salary: undefined']  //7 seconds later  (1) ['Jimmy Quatro, Age: 27 CreditCardNumber: undefined']  Method getEmployees called successfully  (3) ['Sarah Connor, Birthday: undefined Salary: undefined', 'Arnold Schwarzenegger, Birthday: undefined Salary: undefined', 'Kyle Reese, Birthday: undefined Salary: undefined']  //15 seconds later  (0) []  Method getEmployees called successfully  (1) ['Kyle Reese, Birthday: undefined Salary: undefined'] |

|  |
| --- |
| **Input** |
| let userCensorService = new MockCensorService<**User**>([**'age'**])  let employeeCensorService = new MockCensorService<**Employee**>([**'salary'**])  …  const user1 = new User('John Does', 30, 'ABCD-1234');  const user2 = new User('Benny Tres', 23, 'EFGH-5678');  const emp1 = new Employee('Sarah Connor', new Date(1964, 4, 15), 2500);  const emp2 = new Employee('Arnold Schwarzenegger', new Date(1947, 6, 30), 3500);  let usersService = new UsersService([user1, user2], [emp1, emp2]);  let users = usersService.getUsers();  console.log(users.map(x => x.getInfo()));  let employees = usersService.getEmployees();  console.log(employees.map(x => x.getInfo()));  //7 seconds later  setTimeout(() => {      const user3 = new User('Jimmy Quatro', 27, 'IJKL-9012');      const emp3 = new Employee('Kyle Reese', new Date(2004, 0, 1), 2000);      usersService.addUser(user3);      usersService.addEmployee(emp3);      let users = usersService.getUsers();      console.log(users.map(x => x.getInfo()));      let employees = usersService.getEmployees();      console.log(employees.map(x => x.getInfo()));  }, 7000)  //15 seconds later  setTimeout(() => {      let users = usersService.getUsers();      console.log(users.map(x => x.getInfo()));      let employees = usersService.getEmployees();      console.log(employees.map(x => x.getInfo()));  }, 15000) |
| **Output** |
| (2) ['John Does, Age: undefined CreditCardNumber: ABCD-1234', 'Benny Tres, Age: undefined CreditCardNumber: EFGH-5678']  Method getEmployees called successfully  (2) ['Sarah Connor, Birthday: 5/15/1964 Salary: undefined', 'Arnold Schwarzenegger, Birthday: 7/30/1947 Salary: undefined']  //7 seconds later  (1) ['Jimmy Quatro, Age: undefined CreditCardNumber: IJKL-9012']  Method getEmployees called successfully  (3) ['Sarah Connor, Birthday: 5/15/1964 Salary: undefined', 'Arnold Schwarzenegger, Birthday: 7/30/1947 Salary: undefined', 'Kyle Reese, Birthday: 1/1/2004 Salary: undefined']  //15 seconds later  (0) []  Method getEmployees called successfully  (1) ['Kyle Reese, Birthday: 1/1/2004 Salary: undefined'] |