Advanced

June, 10-11 2017

DEVELOPER	13 YEARS
TRAINER	4 YEARS
WRITER	3 BOOKS

FOUNDER

IT DISCOVERY

ITSimulator

IT DISCOVERY

SPEAKER

JAVA DAY
MINSK 2013

polish jug

Dev(Talks):

JEEConf

JAVA DAY 2015

IT talk

Java frameworks day

Angular 5. Basics

- ✓ Angular 5.x vs 2.x vs 1.x
- ✓ TypeScript 2.x
- ✓ Components creation
- ✓ Services and DI
- ✓ Data binding
- ✓ Pipes
- ✓ Form validation
- ✓ Directives
- ✓ Working with HTTP
- ✓ Unit-testing

Sergey Morenets, 2018

Agenda

- ✓ Angular 5 features
- ✓ Yarn
- ✓ Angular Material 2
- ✓ SPA applications
- ✓ Angular Universal
- ✓ Optimization. AOT vs JIT
- ✓ Jest
- ✓ Module development
- ✓ Service workers
- ✓ Angular 6 features

APPROVED

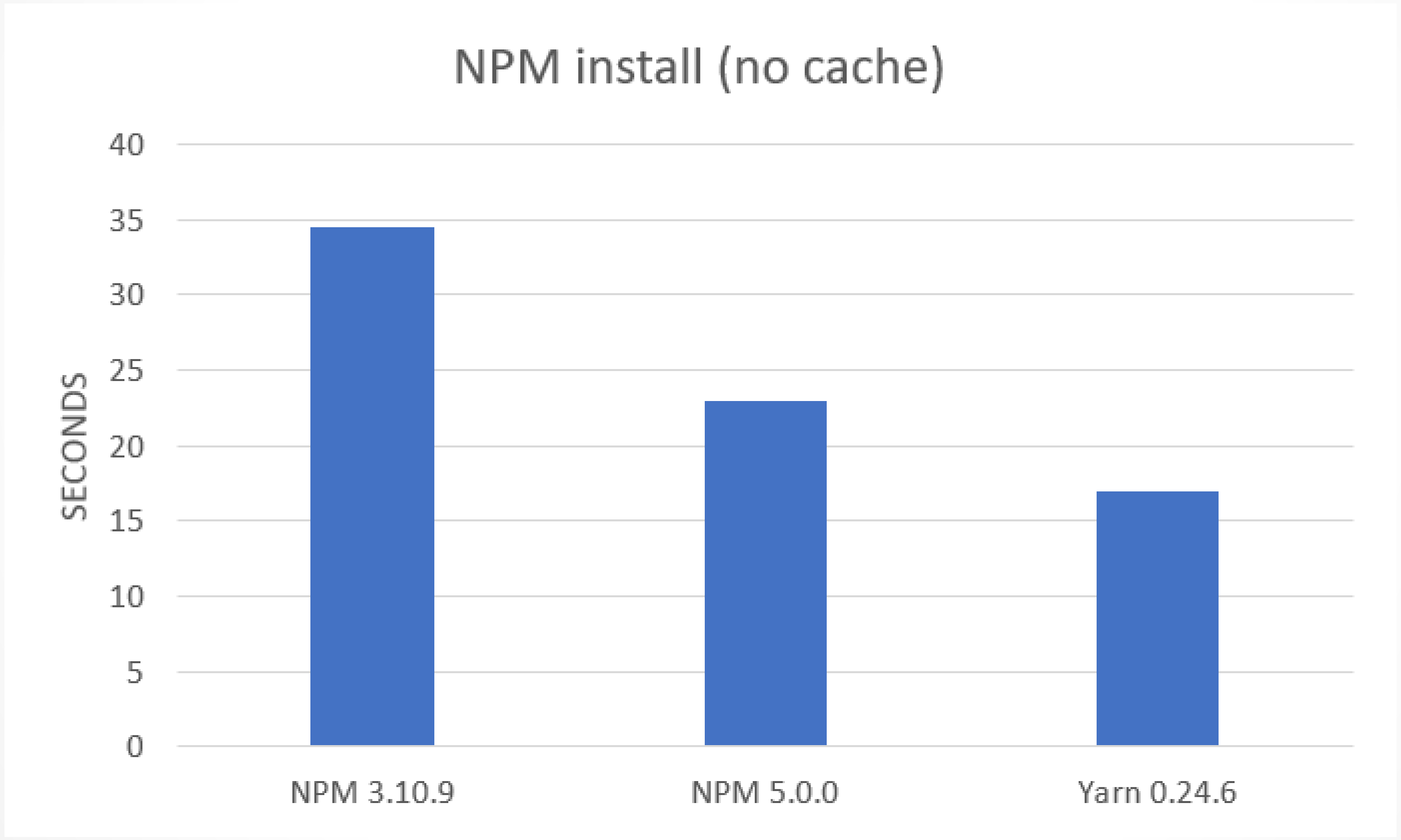
NPM

- ✓ Package manager for JavaScript
- ✓ Bundled together with **Node**
- ✓ Package(or module) is directory with files
- ✓ Hosts over 250 000 packages
- ✓ Manifest is **package.json**

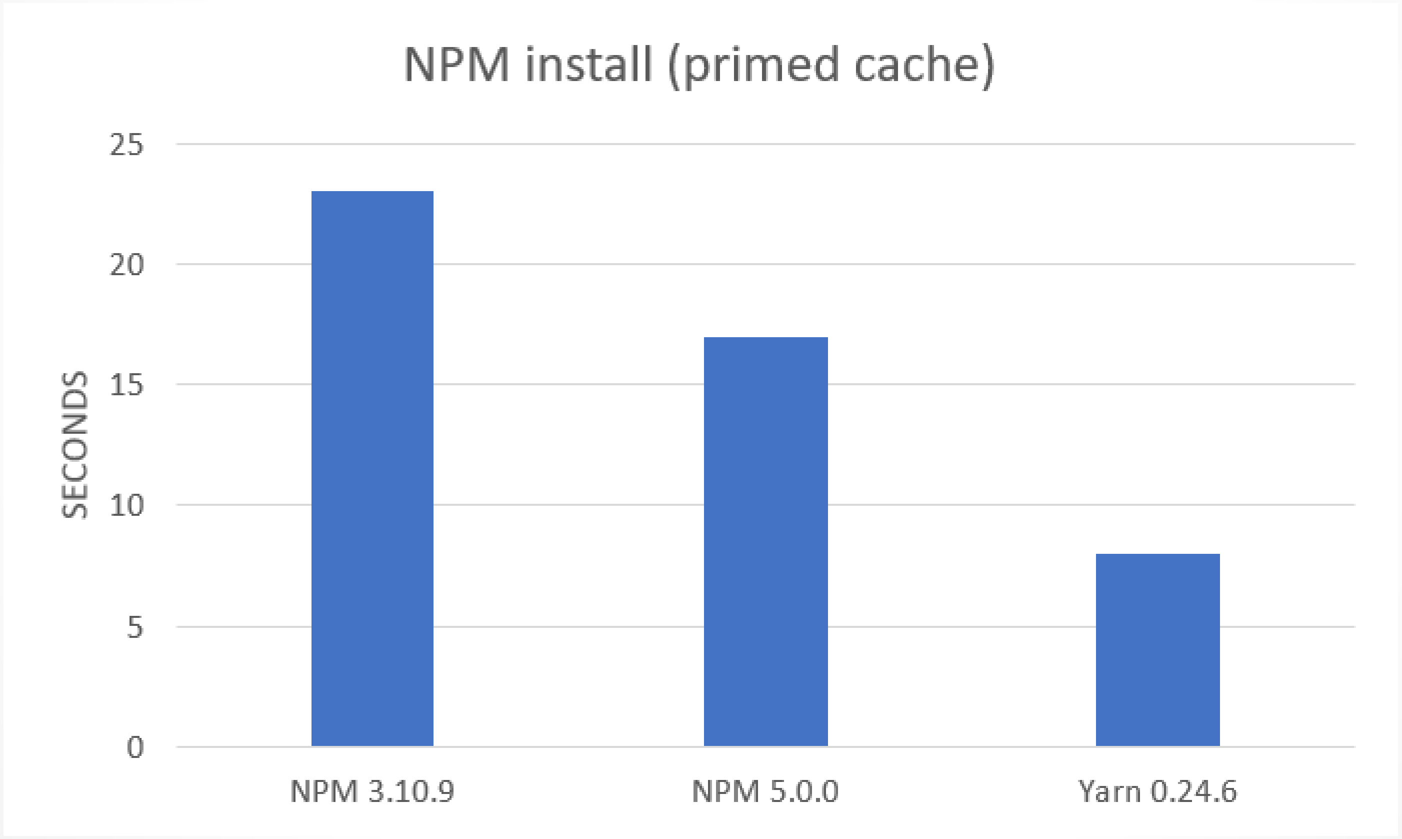
Yarn

- ✓ Built by Facebook, Google, Exponent and Tilde
- ✓ Fetches modules from **NPM** registry
- ✓ Still uses **package.json** for configuration
- ✓ **Parallel** execution(comparing to sequential in NPM)
- ✓ Offline mode
- ✓ Flat mode

Yarn vs NPM


NPM install (no cache)
SECONDS
40
35
30
25
20
15
10
5
0
NPM 3.10.9
NPM 5.0.0
Yarn 0.24.6

Yarn vs NPM


NPM install (primed cache)
SECONDS
25
20
15
10
5
0
NPM 3.10.9
NPM 5.0.0
Yarn 0.24.6

Yarn

- ✓ yarn **init**

    Create empty project

- ✓ yarn [global] **add** <package>@<version> [--dev]

    Adds new dependency

- ✓ yarn **remove** <package>

    Removes dependency

- ✓ yarn **install** or **yarn**

    Installs project dependencies

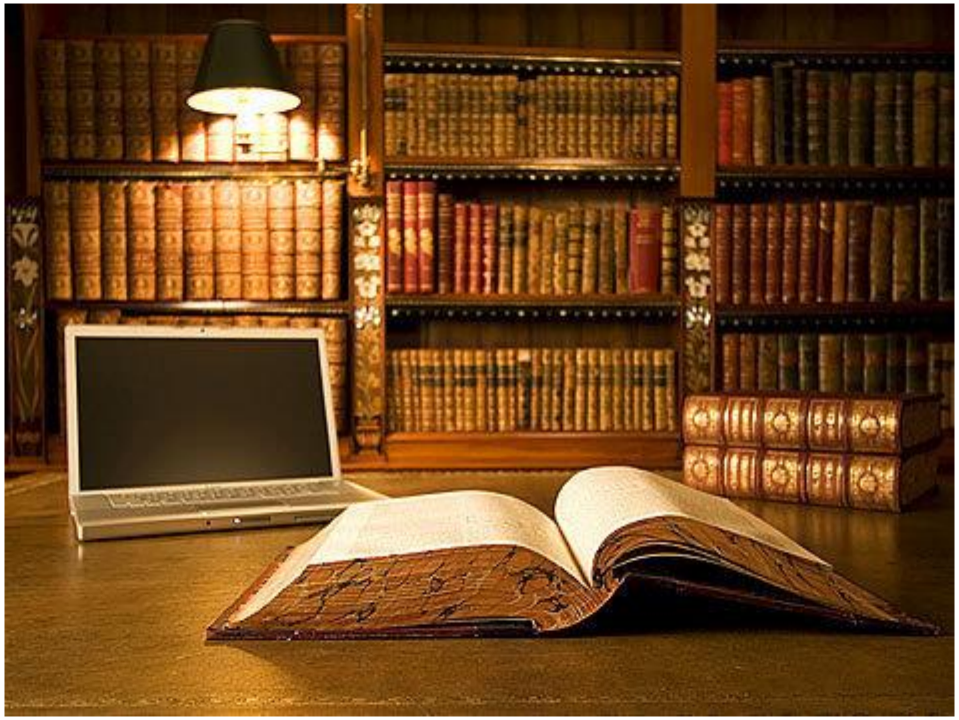
- ✓ *yarn **upgrade** <package>@<version>*

    Upgrades dependency

Task #1. Installation.

1. Install **Node**/npm.Check **NPM** version using “npm –v” command.
2. Install **Python**(2.7 or higher)
3. Install **Git**
4. Install **Yarn**. You can install it using NPM: *npm install yarn --global* . However native OS manager is recommended:
5. Create **new folder** and run *yarn init* command in this folder.

Business domain



Task #2. Project installation.

1. Clone project from Git repository: https://github.com/it-discovery/angular
2. Install all the required dependencies: **yarn**.
3. Open project in IDE (**WebStorm** 2018.1 is recommended)
4. Review project components, services, directives.
5. Start project: **ng serve** and observe its functionality

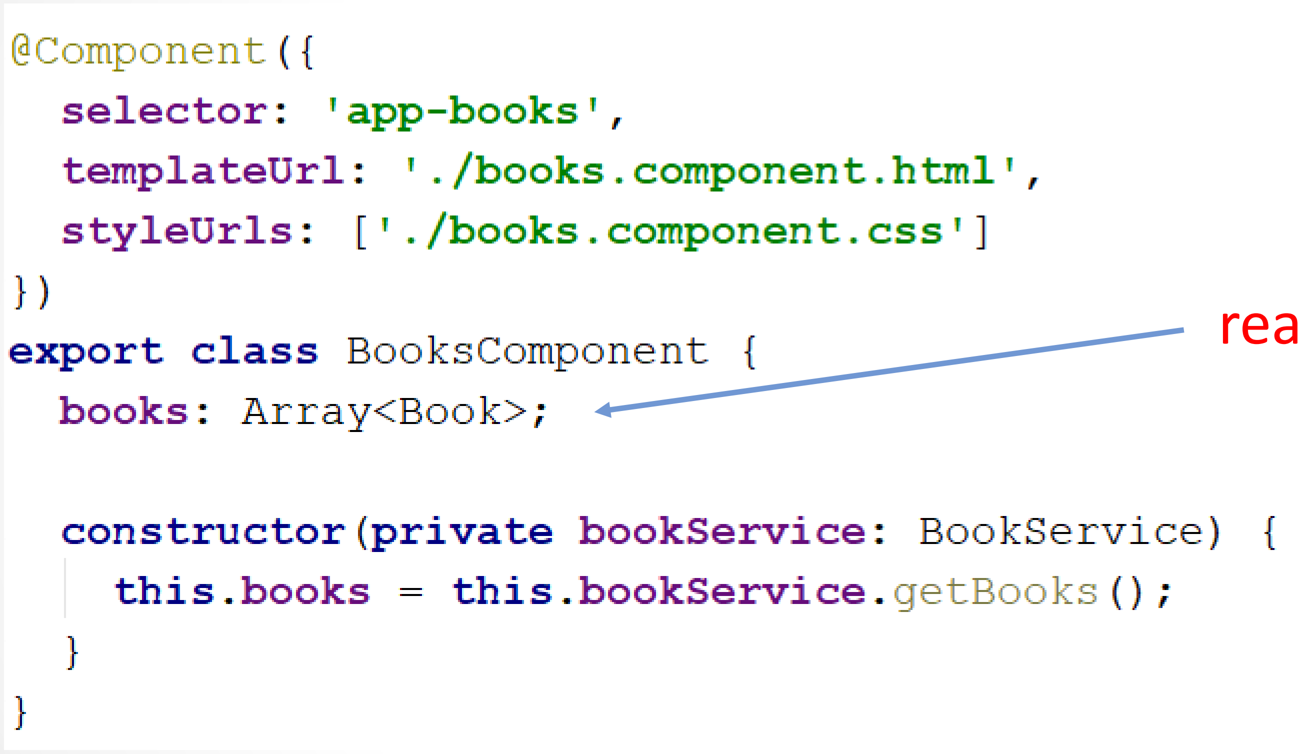
Task #3. Data binding.

1. Create new component with **<button>** and **<div>** elements. For example: *<div id="my_div"></div><button>Update</button>*
2. Create all possible approaches to change content of **<div>** element by clicking 'Update' button using **Angular**.

Common mistakes/flaws

0-11 2017

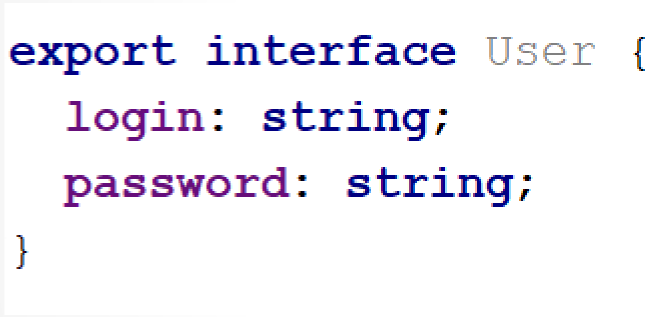
Common mistakes/flaws



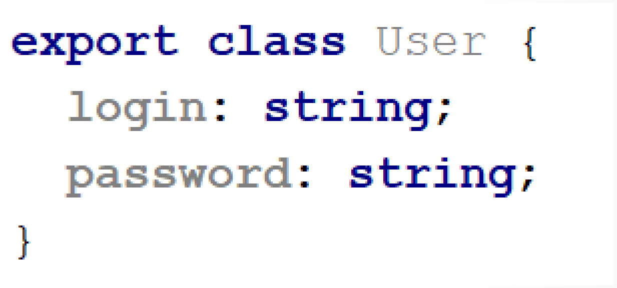
```
@Component({
  selector: 'app-books',
  templateUrl: './books.component.html',
  styleUrls: ['./books.component.css']
})
export class BooksComponent {
  books: Array<Book>;

  constructor(private bookService: BookService) {
    this.books = this.bookService.getBooks();
  }
}
```

readonly

Interfaces vs classes

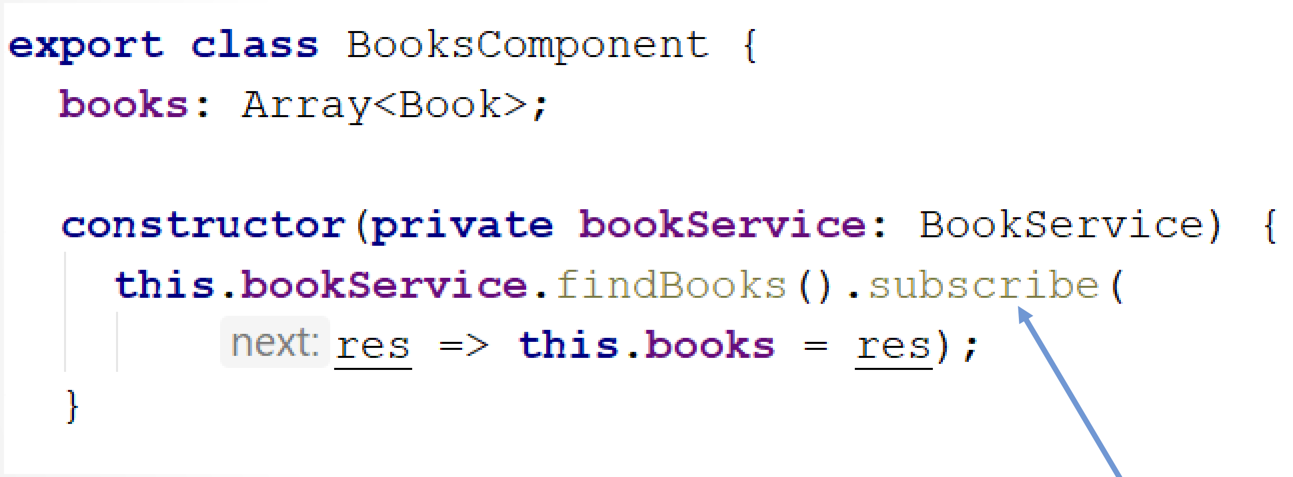


```
export interface User {
  login: string;
  password: string;
}
```



```
export class User {
  login: string;
  password: string;
}
```

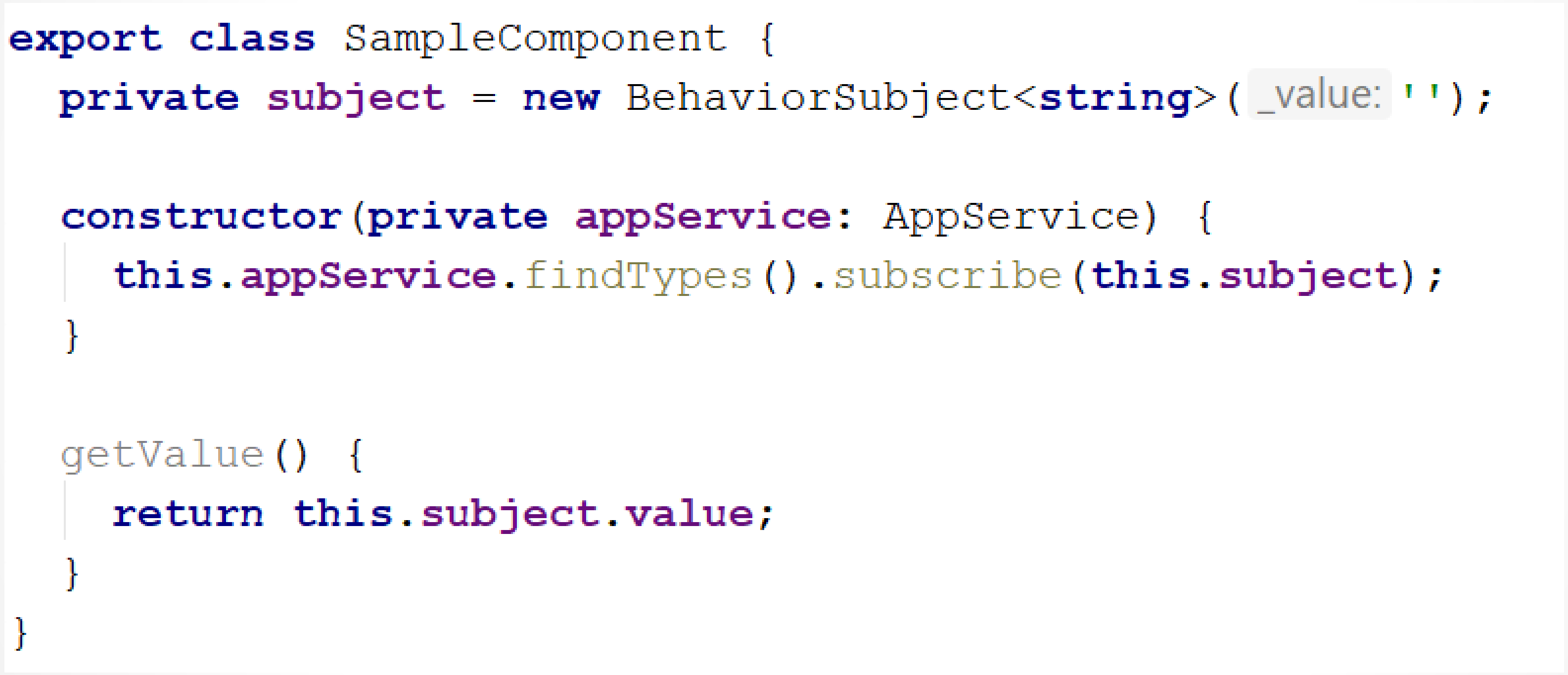
Common mistakes/flaws



```
export class BooksComponent {
  books: Array<Book>;

  constructor(private bookService: BookService) {
    this.bookService.findBooks().subscribe(
          next: res => this.books = res);
  }
```

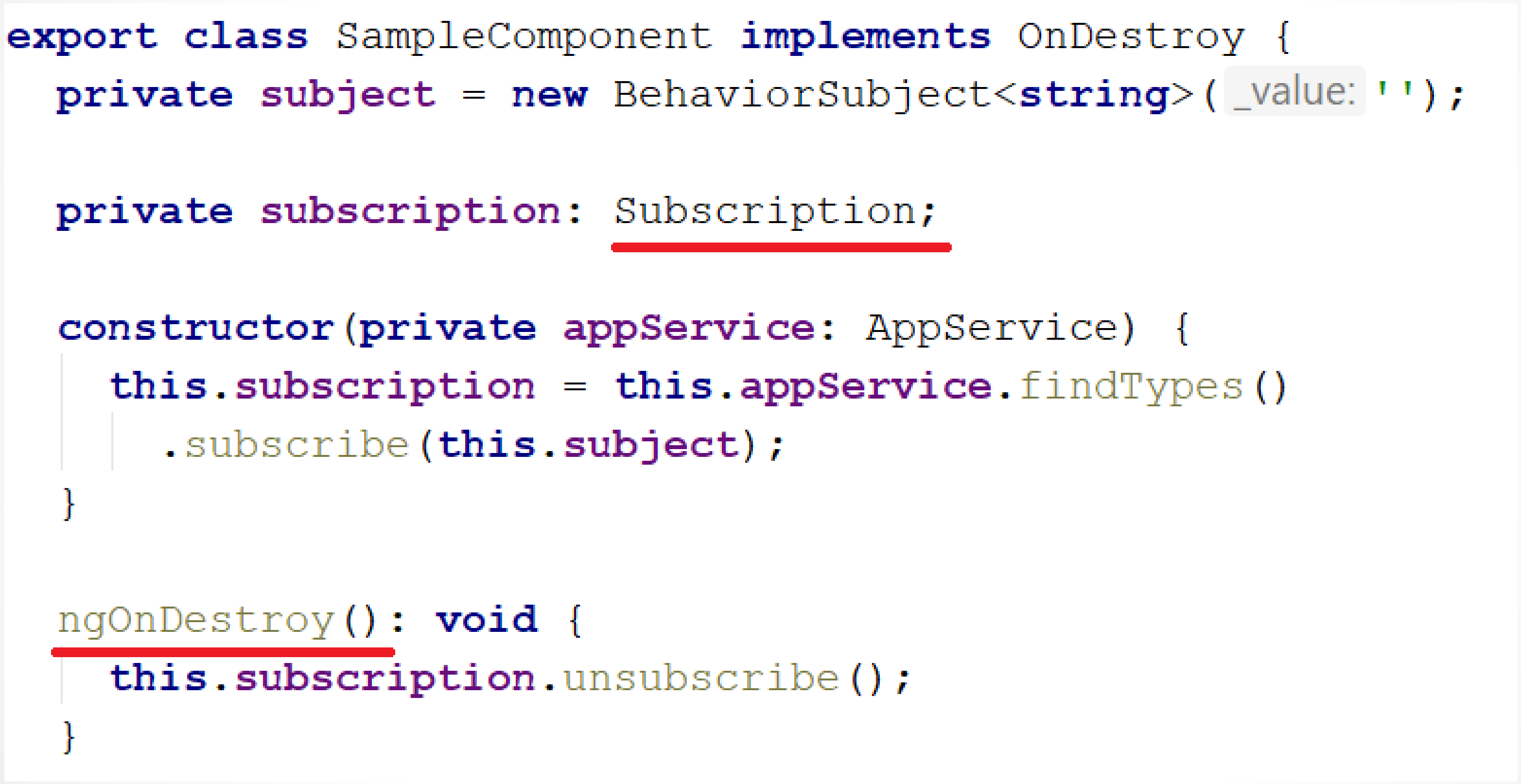
ngOnInit

Common mistakes/flaws



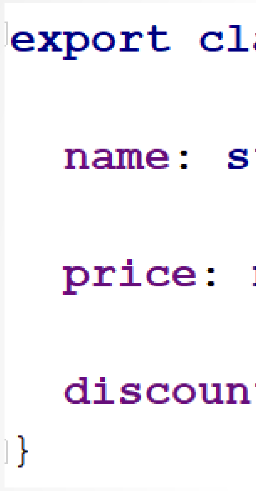
```
export class SampleComponent {
  private subject = new BehaviorSubject<string>( _value: '');

  constructor(private appService: AppService) {
    this.appService.findTypes().subscribe(this.subject);
  }

  getValue() {
    return this.subject.value;
  }
}
```

Common mistakes/flaws

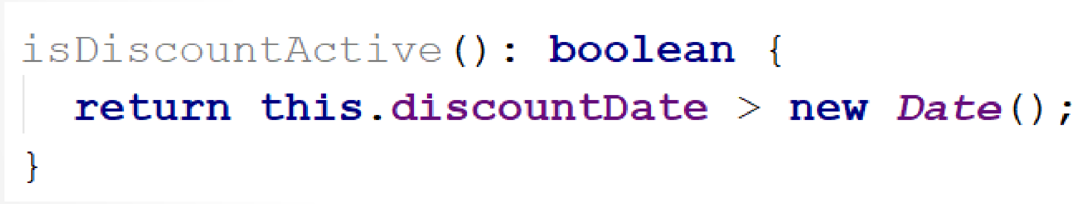


```
export class SampleComponent implements OnDestroy {
  private subject = new BehaviorSubject<string>( _value: '');

  private subscription: Subscription;

  constructor(private appService: AppService) {
    this.subscription = this.appService.findTypes()
      .subscribe(this.subject);
  }

  ngOnDestroy(): void {
    this.subscription.unsubscribe();
  }
```

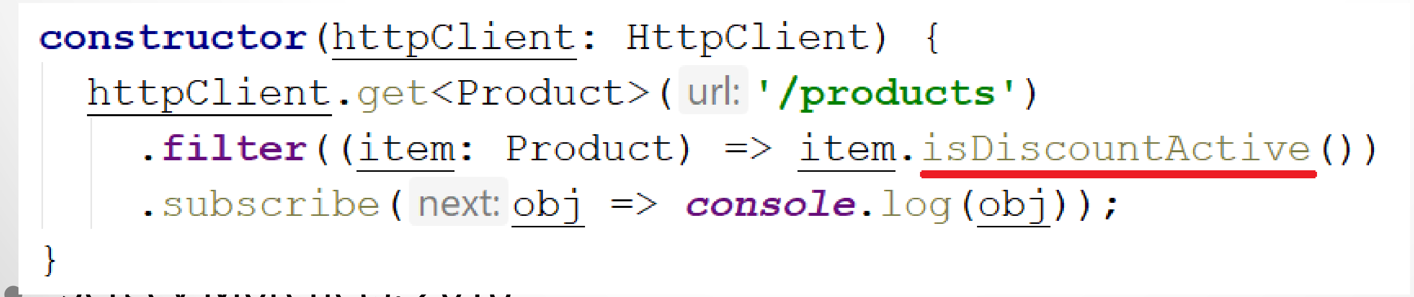
Common mistakes/flaws



```
export class Product {

  name: string;

  price: number;

  discountDate: Date;
}
```



```
isDiscountActive(): boolean {
  return this.discountDate > new Date();
}
```



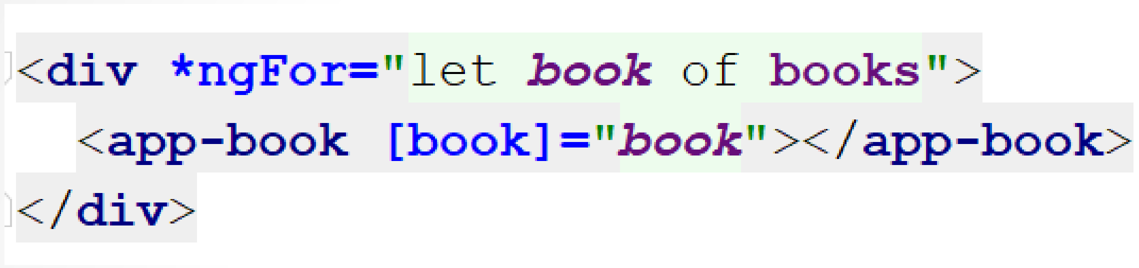
```
constructor(httpClient: HttpClient) {
  httpClient.get<Product>( url: '/products')
    .filter((item: Product) => item.isDiscountActive())
    .subscribe( next: obj => console.log(obj));
}
```

Common mistakes/flaws

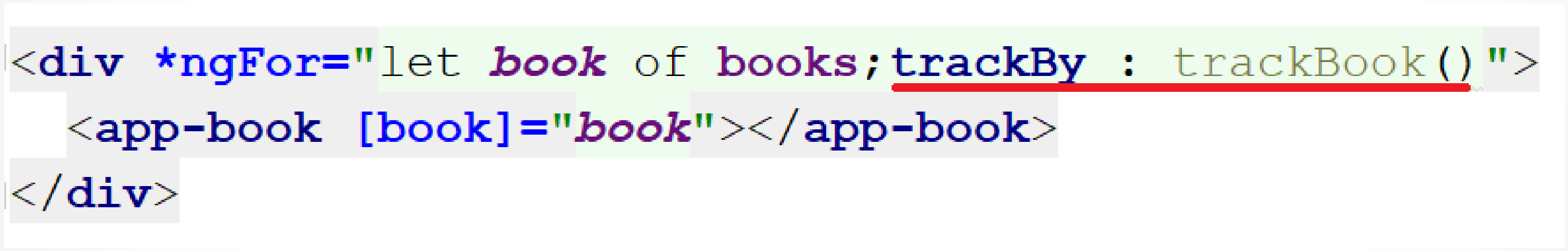
```
export class DataComponent implements OnInit {

  constructor(private activeModal: NgbActiveModal) {
  }
```

```
constructor(private el: ElementRef, private service: HighlightJsService,
            private route: ActivatedRoute,private articleService: ArticleService,
            authService: AuthenticationService,
            private subscribeService: SubscribeService,
            private attachmentService: AttachmentService) {
  super(authService);
```

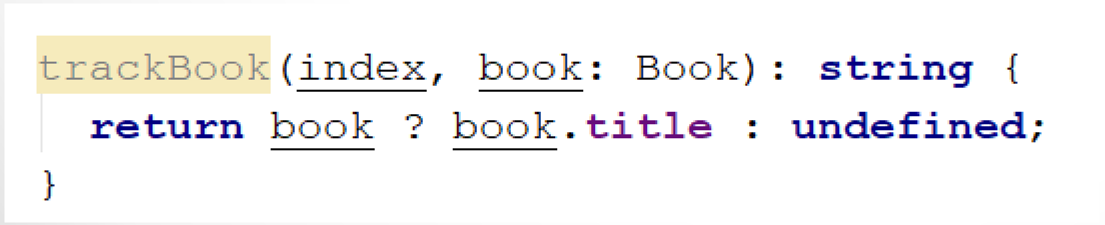
Common mistakes/flaws



```
<div *ngFor="let book of books">
  <app-book [book]="book"></app-book>
</div>
```



```
<div *ngFor="let book of books;trackBy : trackBook()">
  <app-book [book]="book"></app-book>
</div>
```



```
trackBook(index, book: Book): string {
  return book ? book.title : undefined;
}
```

Angular 5 features

- ✓ Service workers
- ✓ Form changes
- ✓ HTTP Client
- ✓ Router changes
- ✓ i18N

Angular 5. Form changes

```
this.passwordCtrl = new FormControl('', {
  validators: Validators.required,
  updateOn: 'blur'
});
```

blur, **change**, submit

```
this.userForm = new FormGroup({
  username: '',
  password: ''
}, { updateOn: 'blur' });
```

Angular 5. HTTP Client

```
const headers = new HttpHeaders().set('Authorization', '
secret');
const params = new HttpParams().set('page', '1');
return this.http.get('/api/users', { headers, params });
```

Angular 4.3

```
const headers = { 'Authorization': 'secret' };
const params = { 'page': '1' };
return this.http.get('/api/users', { headers, params });
```

Angular 5.x

Router changes

- ✓ ChildActivationStart and ChildActivationEnd events
- ✓ Possibility to reload a route

```
providers: [
  // ...
  RouterModule.forRoot(routes, {
    onSameUrlNavigation: 'reload'
  })
]
```

i18 changes

```
import { registerLocaleData } from '@angular/common';
import localeFr from '@angular/common/locales/fr';

registerLocaleData(localeFr);
```

```
@Component({
  selector: 'ns-locale',
  template: `
    <p>The locale is {{ locale }}</p>
    <!-- will display 'en-US' -->

    <p>{{ 1234.56 | number:'1.0-3':'fr-FR' }}</p>
    <!-- will display '1 234,56' -->
  `
})
class DefaultLocaleComponentOverridden {
  constructor(@Inject(LOCALE_ID) public locale: string) { }
```

Pipes

- ✓ Used to transform and filter data
- ✓ Raw data formatting
- ✓ Called ‘filters’ in Angular 1.x
- ✓ Can be used in HTML/application code

Built-in pipes

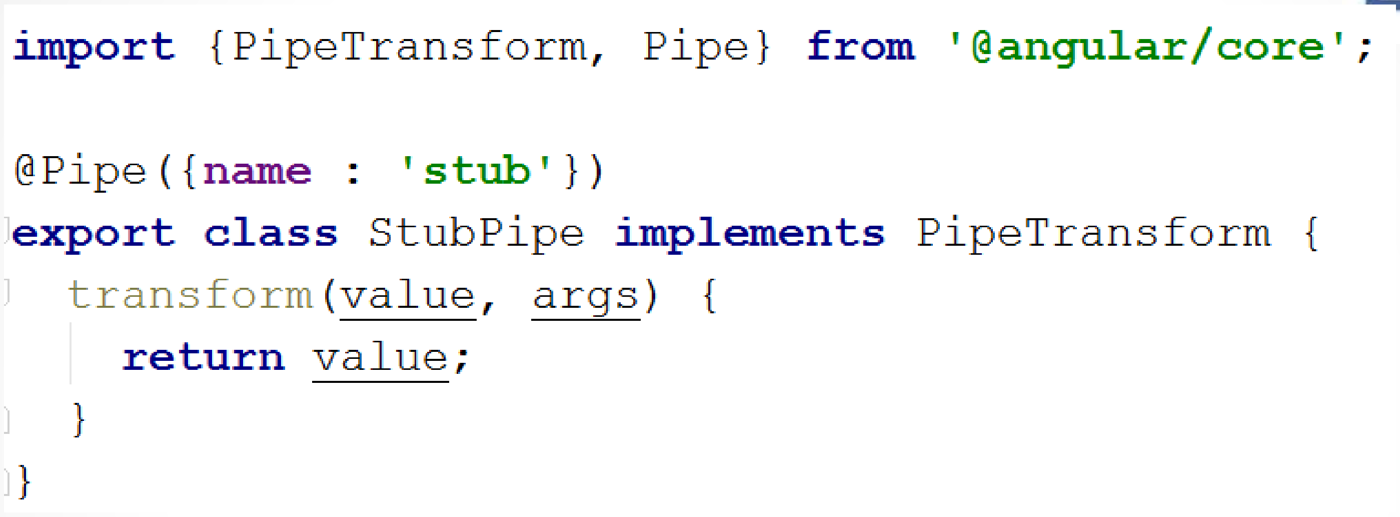
Name	Description	Example
json	Converts object into JSON text fomat	{{book | **json**}}
slice	Filters collection(array) creating new sub-collection	*ngFor="let book of books | **slice** : 0: 5"

number	Formats number using current regional settings	{{ amount | **number** }}
		{{ amount | **number** : ‘.2-2’}}

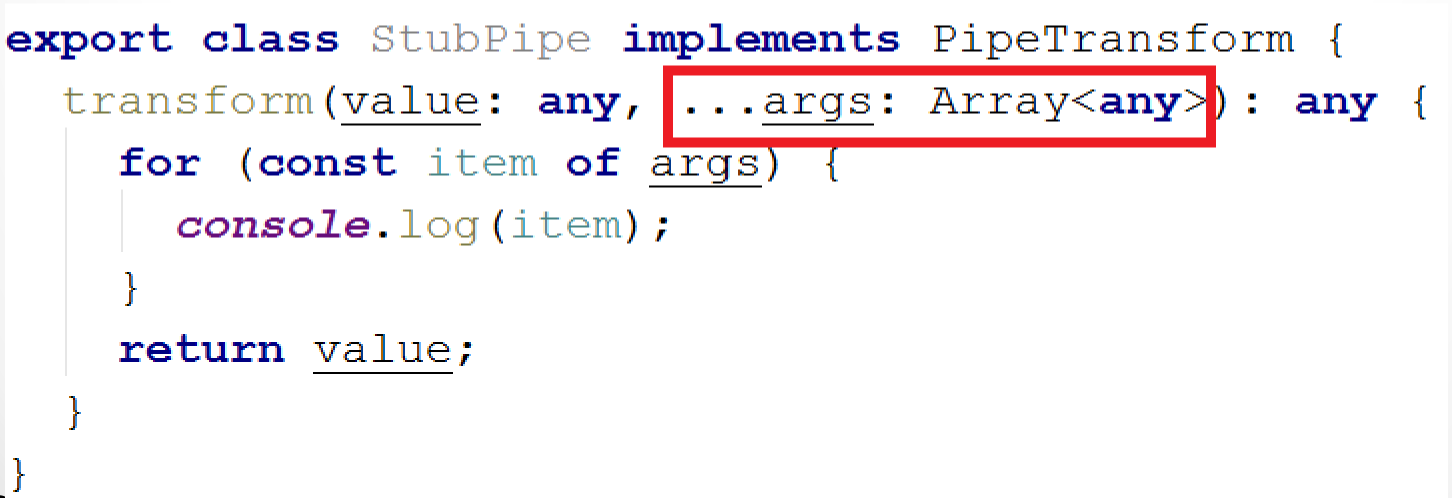
percent	Format number into percentage format	{{ amount | **percent** }}

currency	Applies currency symbol to the number	{{ amount | **currency** }}
		{{ amount | **currency** : ‘UAH’}}
		{{ amount | **currency** : ‘EUR’ : true}}

User-defined pipes



```
import {PipeTransform, Pipe} from '@angular/core';

@Pipe({name : 'stub'})
export class StubPipe implements PipeTransform {
  transform(value, args) {
    return value;
  }
}
```



```
export class StubPipe implements PipeTransform {
  transform(value: any, ...args: Array<any>): any {
    for (const item of args) {
      console.log(item);
    }
    return value;
  }
}
```

Task #4. Custom pipe

1. Create new pipe “**sort**” that will sort elements of array (ascending or descending). The format of this pipe will be the following:
2. items | sort : ‘title’ -- sorts **items** array by **title** property in ascending order
3. Create new ‘**ui**’ module and put this pipe there.
4. Apply this pipe to the **BooksComponent** to sort books by title or author.

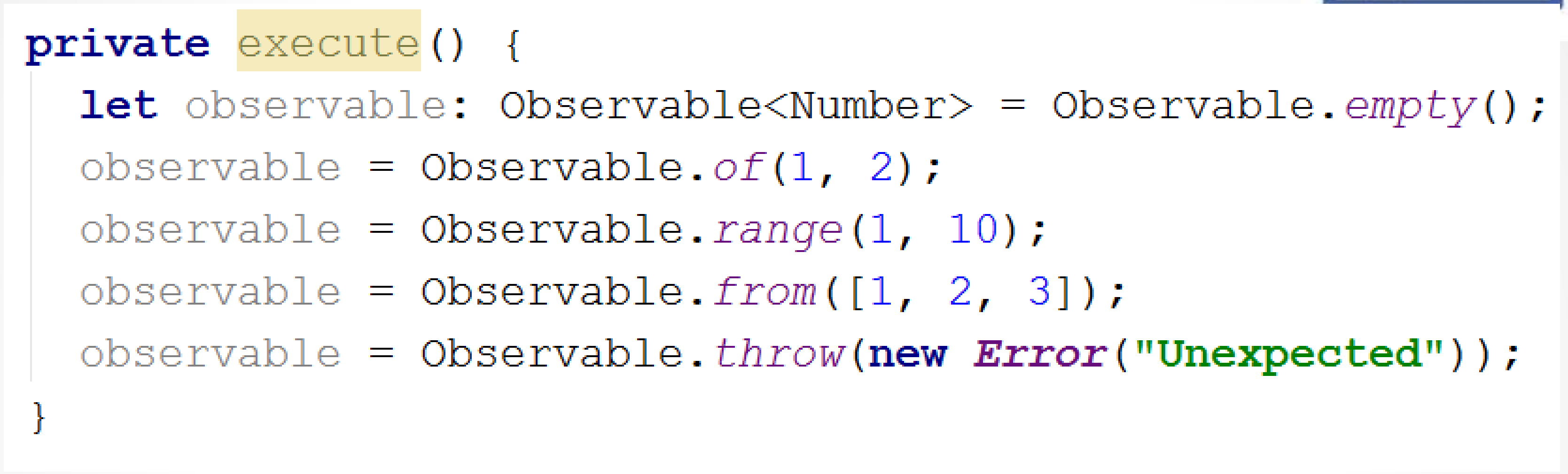
RxJS

- ✓ Push-based collection of events
- ✓ Implementation of observer design pattern
- ✓ Observable object sends notifications
- ✓ Observer object receive notifications

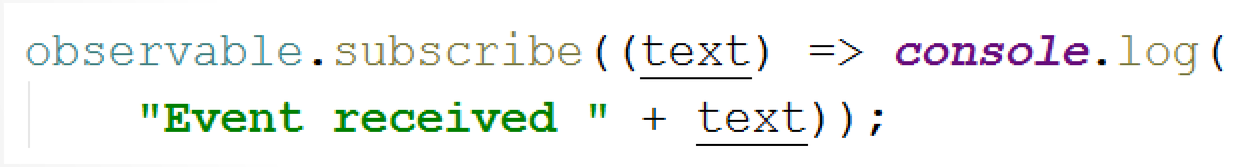
RxJS

Type	Description
Observable	Push-based collection of values
Observer	Consumer of push-based notifications from Observer
Subscription	Result of subscription of Observer on Observable
Scheduler	Allows to schedule/order tasks execution
Subject	Can act as both data producer/consumer

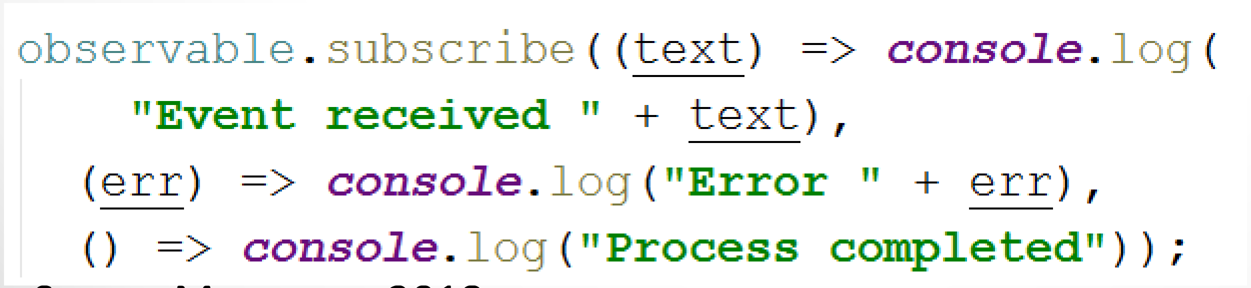
Observable. Creation



```
private execute() {
   let observable: Observable<Number> = Observable.empty();
   observable = Observable.of(1, 2);
   observable = Observable.range(1, 10);
   observable = Observable.from([1, 2, 3]);
   observable = Observable.throw(new Error("Unexpected"));
}
```



```
observable.subscribe((text) => console.log(
     "Event received " + text));
```



```
observable.subscribe((text) => console.log(
     "Event received " + text),
   (err) => console.log("Error " + err),
   () => console.log("Process completed"));
```

Observable. Creation

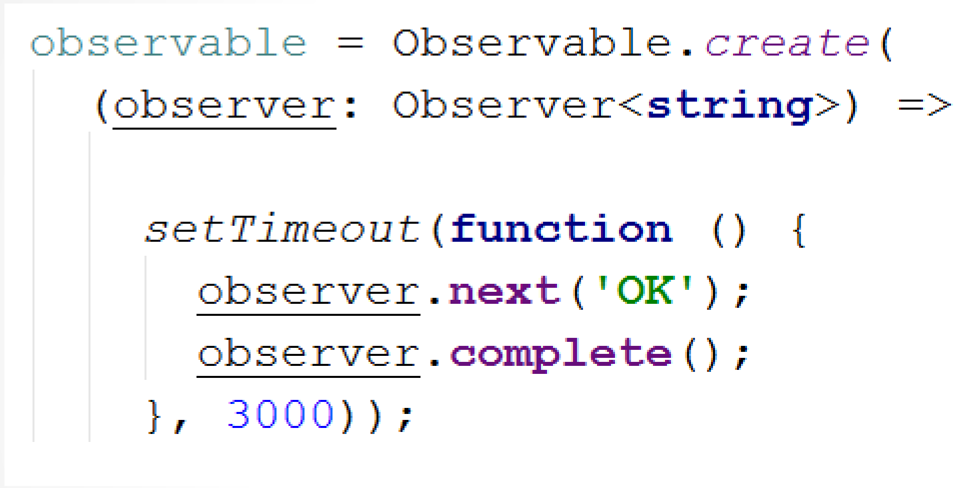
```
observable = Observable.interval(2000);
observable.subscribe(x => console.log(x));
```

0, 1, 2, 3, …

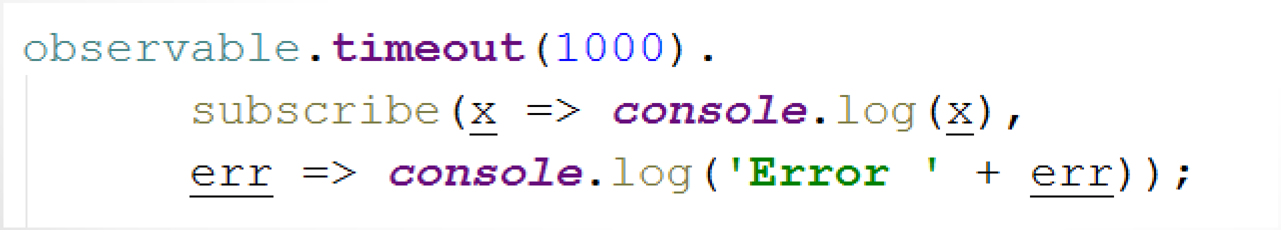
```
const subsription: Subscription = Observable.from([1, 2, 3])
    .subscribe(x => console.log(x));
subsription.unsubscribe();
```

```
const subject: Subject<number> = new Subject();
subject.subscribe(x => console.log(x));
subject.subscribe(x => this.exec(x));

observable.subscribe(subject);
```

Observable. Creation

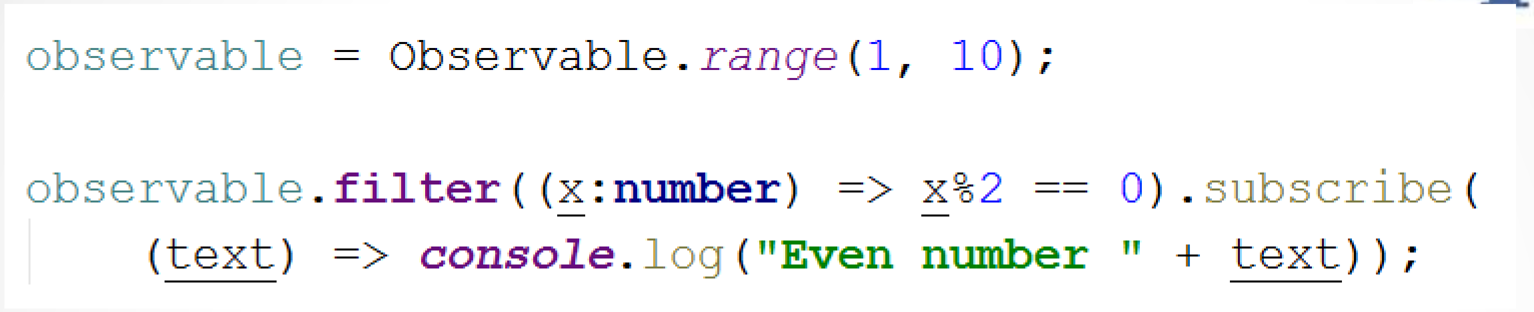


```
observable = Observable.create(
  (observer: Observer<string>) =>

    setTimeout(function () {
      observer.next('OK');
      observer.complete();
    }, 3000));
```

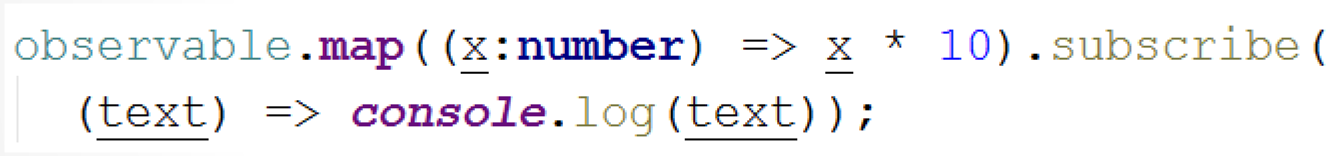


```
observable.timeout(1000).
      subscribe(x => console.log(x),
      err => console.log('Error ' + err));
```

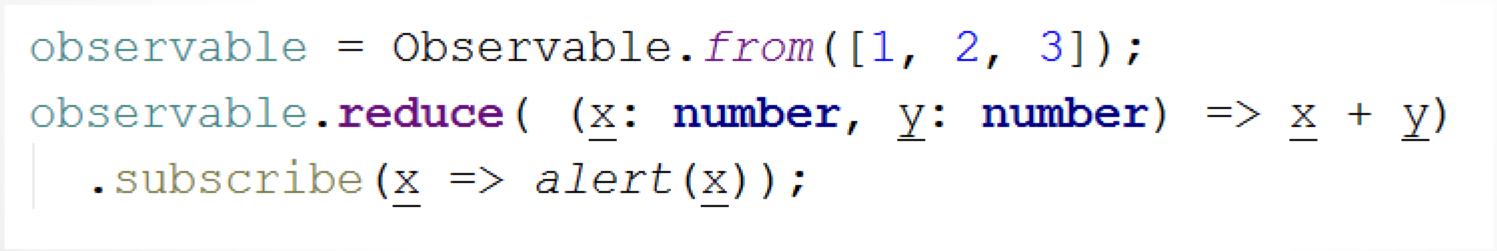
Observable. Operations



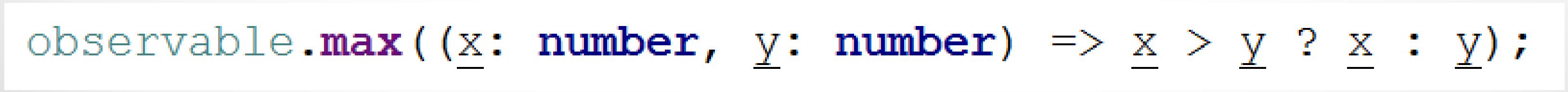
```
observable = Observable.range(1, 10);

observable.filter((x:number) => x%2 == 0).subscribe(
    (text) => console.log("Even number " + text));
```



```
observable.map((x:number) => x * 10).subscribe(
  (text) => console.log(text));
```



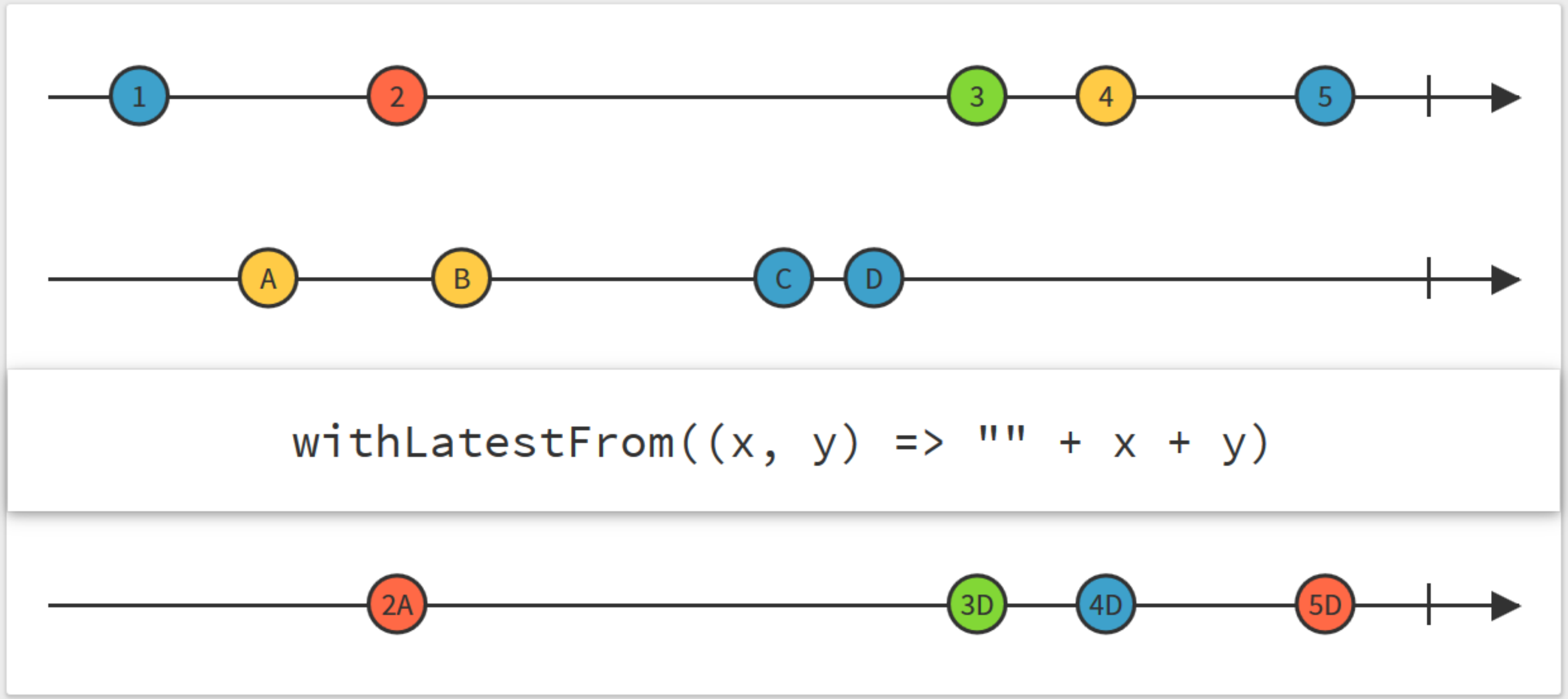
```
observable = Observable.from([1, 2, 3]);
observable.reduce( (x: number, y: number) => x + y)
  .subscribe(x => alert(x));
```



```
observable.max((x: number, y: number) => x > y ? x : y);
```

- Sergey Morenets, 2018

RxMarbles



Sergey Morenets, 2018

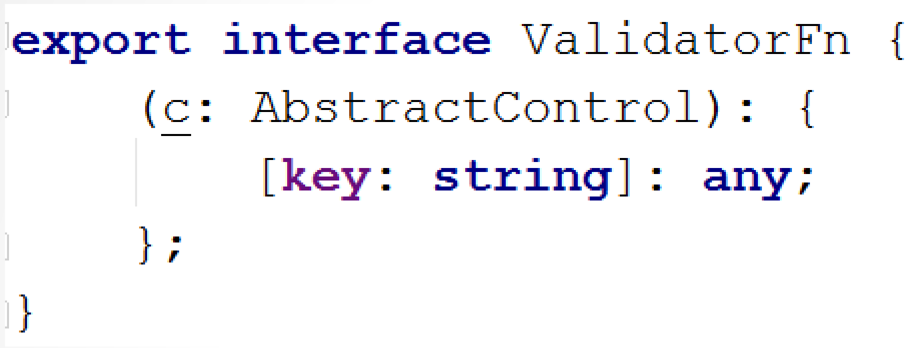
Task #5. Observable

1. Create few Observable objects, using **from(), range()** and **of ()** functions.
2. Subscribe to observable to print received events or generated errors.
3. Create Observable that pushes **Latin** lowercase letters (from ‘a’ to ‘z’) and verify its behavior.

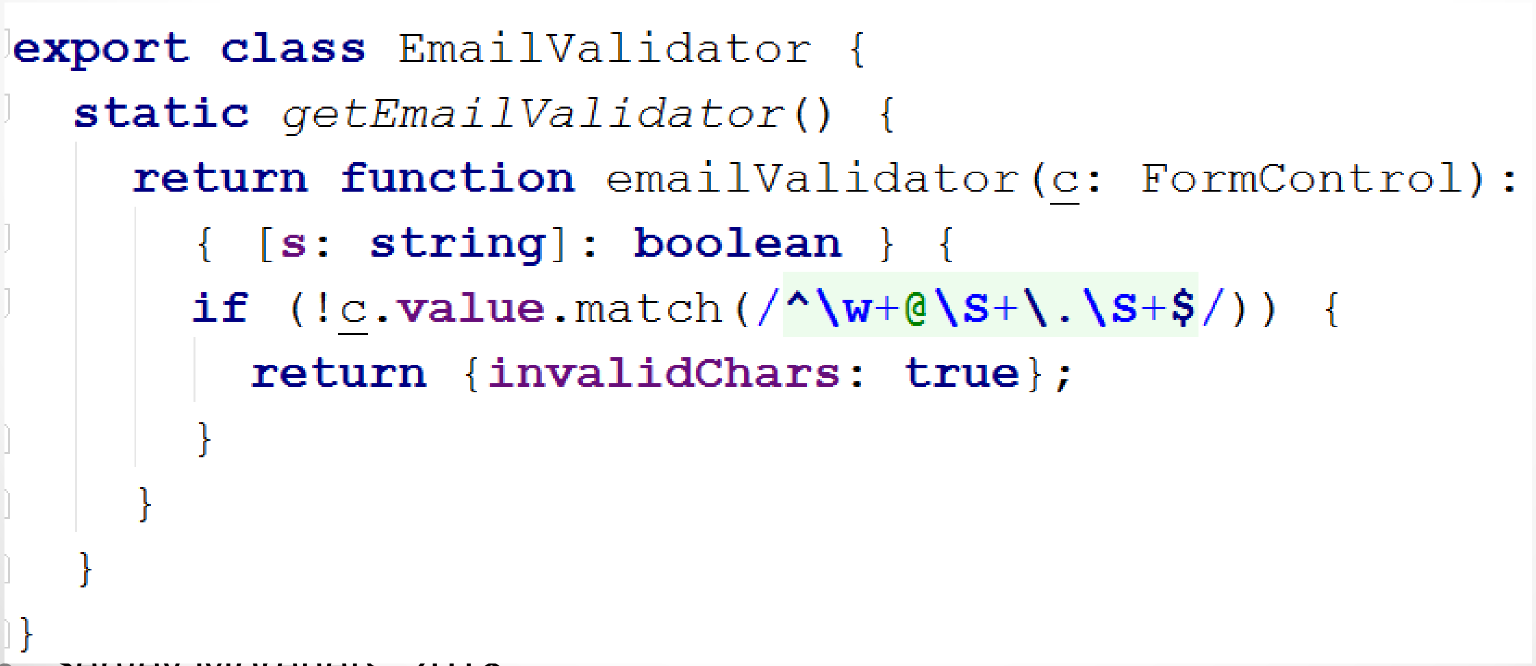
Task #6. Global Event Bus

1. Create new class **ApplicationEvent** with source and message properties.
2. Create new service **AsyncEventBus**. It will serve as global event bus and allow components to:
   1. Send events of **ApplicationEvent** type
   2. Subscribe to **ApplicationEvent** events
3. Use **RxJs** functionality to generate/subscribe for events.

Custom validator

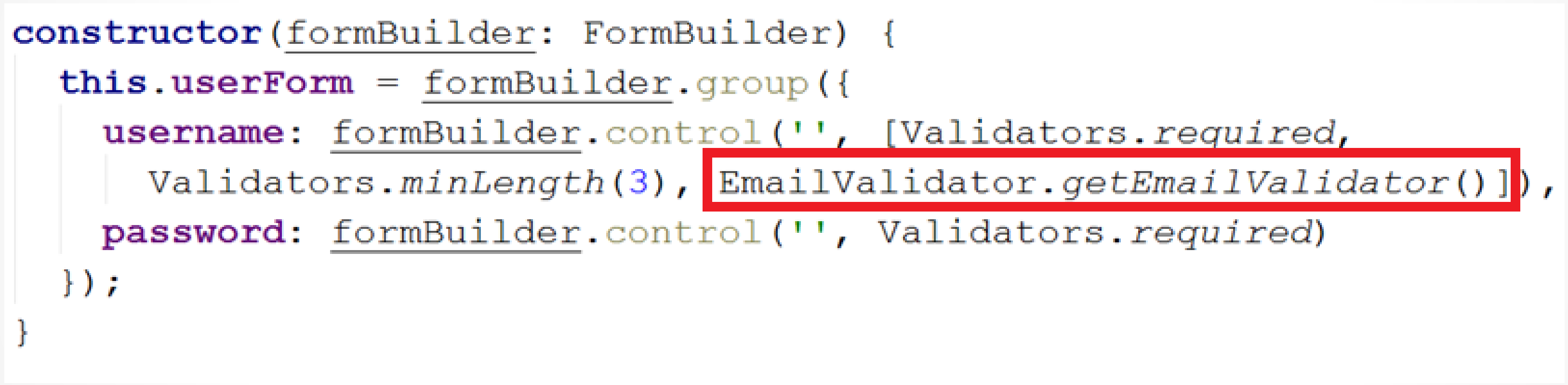


```
export interface ValidatorFn {
    (c: AbstractControl): {
        [key: string]: any;
    };
}
```



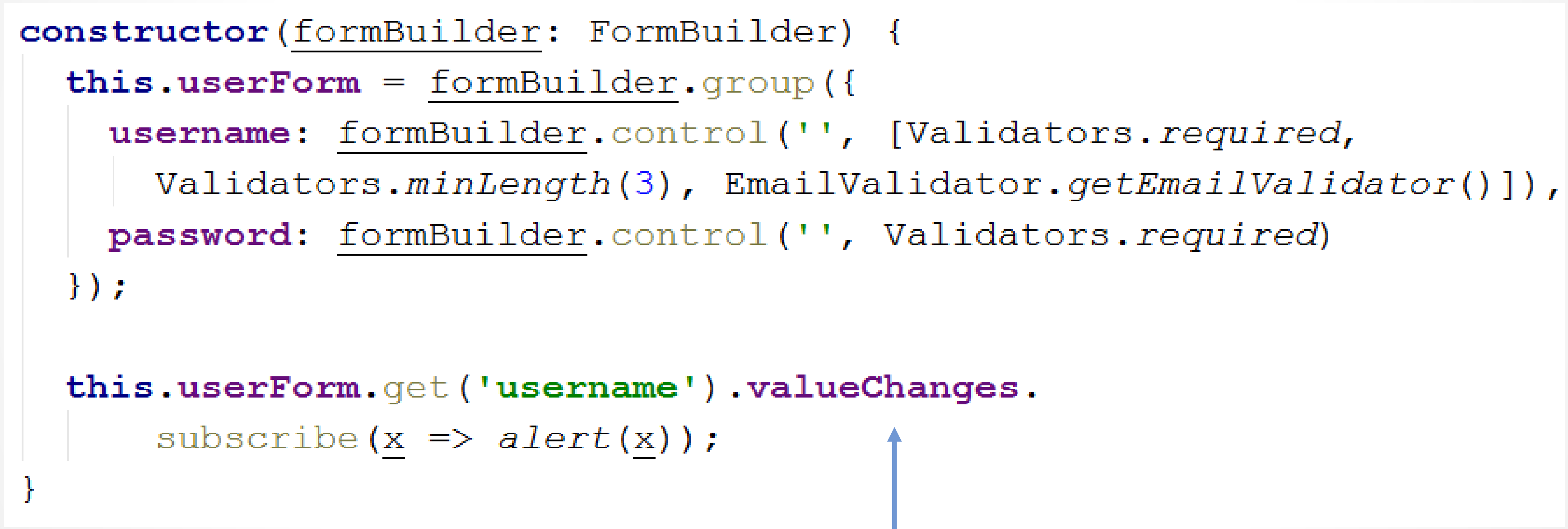
```
export class EmailValidator {
  static getEmailValidator() {
    return function emailValidator(c: FormControl):
      { [s: string]: boolean } {
      if (!c.value.match(/^\w+@\S+\.\S+$/)) {
        return {invalidChars: true};
      }
    }
  }
}
```

Custom validator



```
constructor(formBuilder: FormBuilder) {
  this.userForm = formBuilder.group({
    username: formBuilder.control('', [Validators.required,
      Validators.minLength(3), EmailValidator.getEmailValidator()]),
    password: formBuilder.control('', Validators.required)
  });
}
```

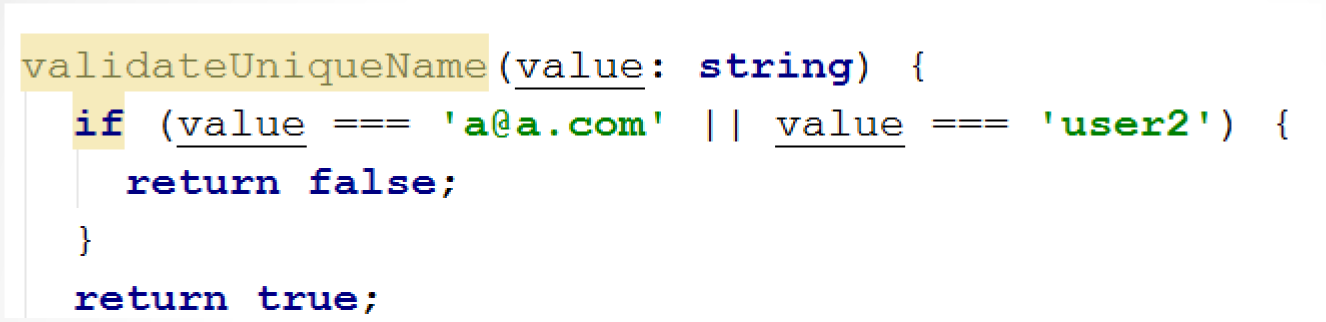
Asynchronous validator



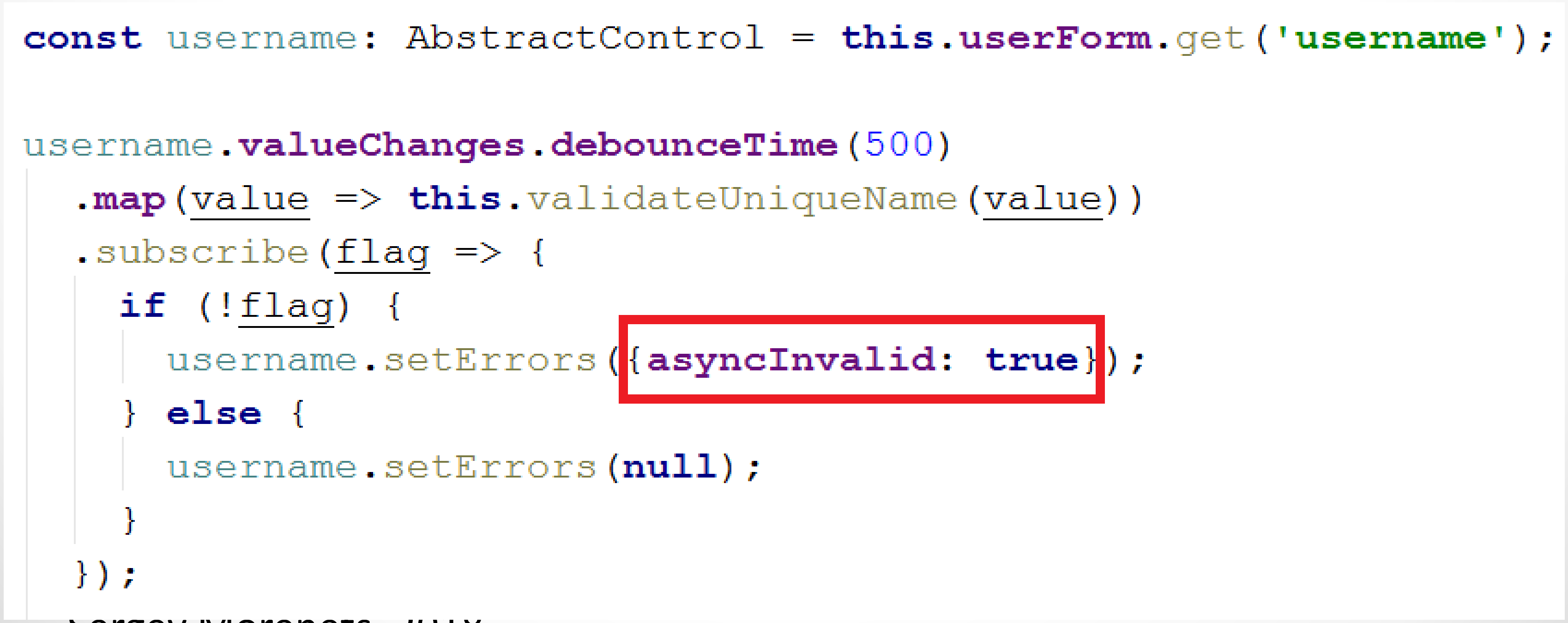
```
constructor(formBuilder: FormBuilder) {
  this.userForm = formBuilder.group({
    username: formBuilder.control('', [Validators.required,
      Validators.minLength(3), EmailValidator.getEmailValidator()]),
    password: formBuilder.control('', Validators.required)
  });

  this.userForm.get('username').valueChanges.
      subscribe(x => alert(x));
}
```

returns **Observable**

Asynchronous validator

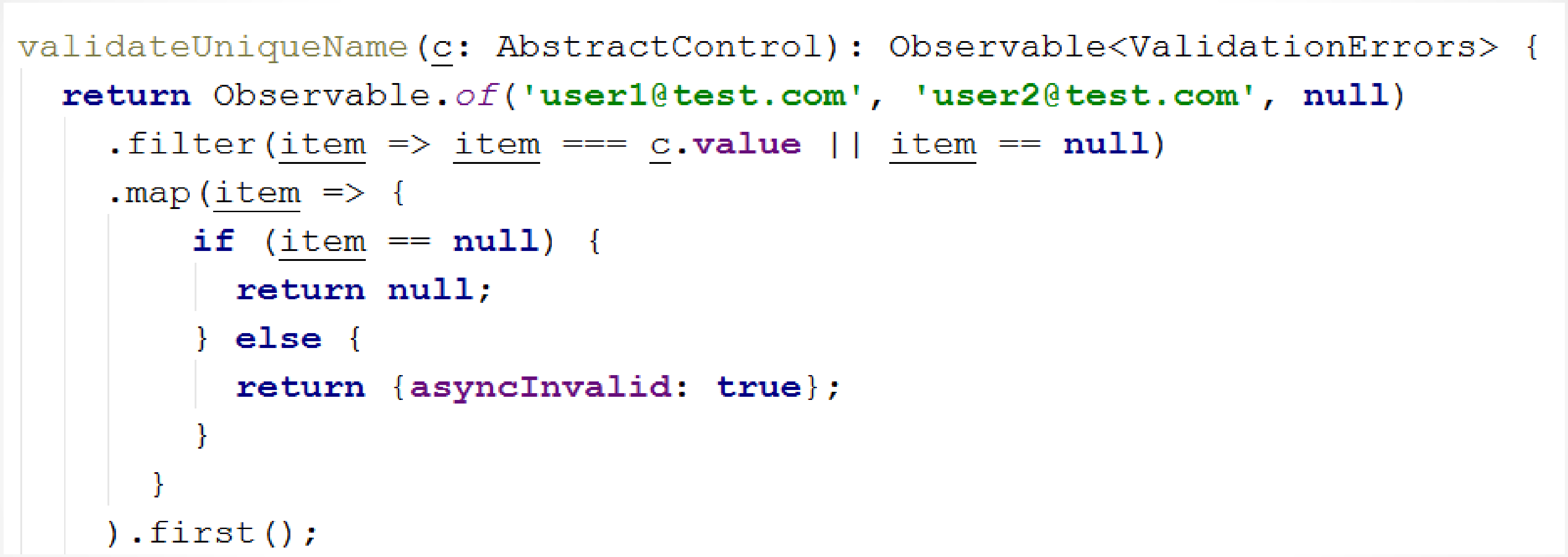


```
validateUniqueName(value: string) {
  if (value === 'a@a.com' || value === 'user2') {
    return false;
  }
  return true;
```

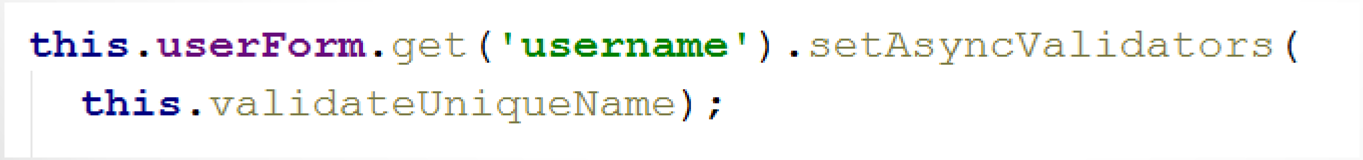


```
const username: AbstractControl = this.userForm.get('username');

username.valueChanges.debounceTime(500)
  .map(value => this.validateUniqueName(value))
  .subscribe(flag => {
    if (!flag) {
      username.setErrors({asyncInvalid: true});
    } else {
      username.setErrors(null);
    }
  });
```

Asynchronous validator



```
validateUniqueName(c: AbstractControl): Observable<ValidationErrors> {
  return Observable.of('user1@test.com', 'user2@test.com', null)
    .filter(item => item === c.value || item == null)
    .map(item => {
        if (item == null) {
          return null;
        } else {
          return {asyncInvalid: true};
        }
      }
    ).first();
```

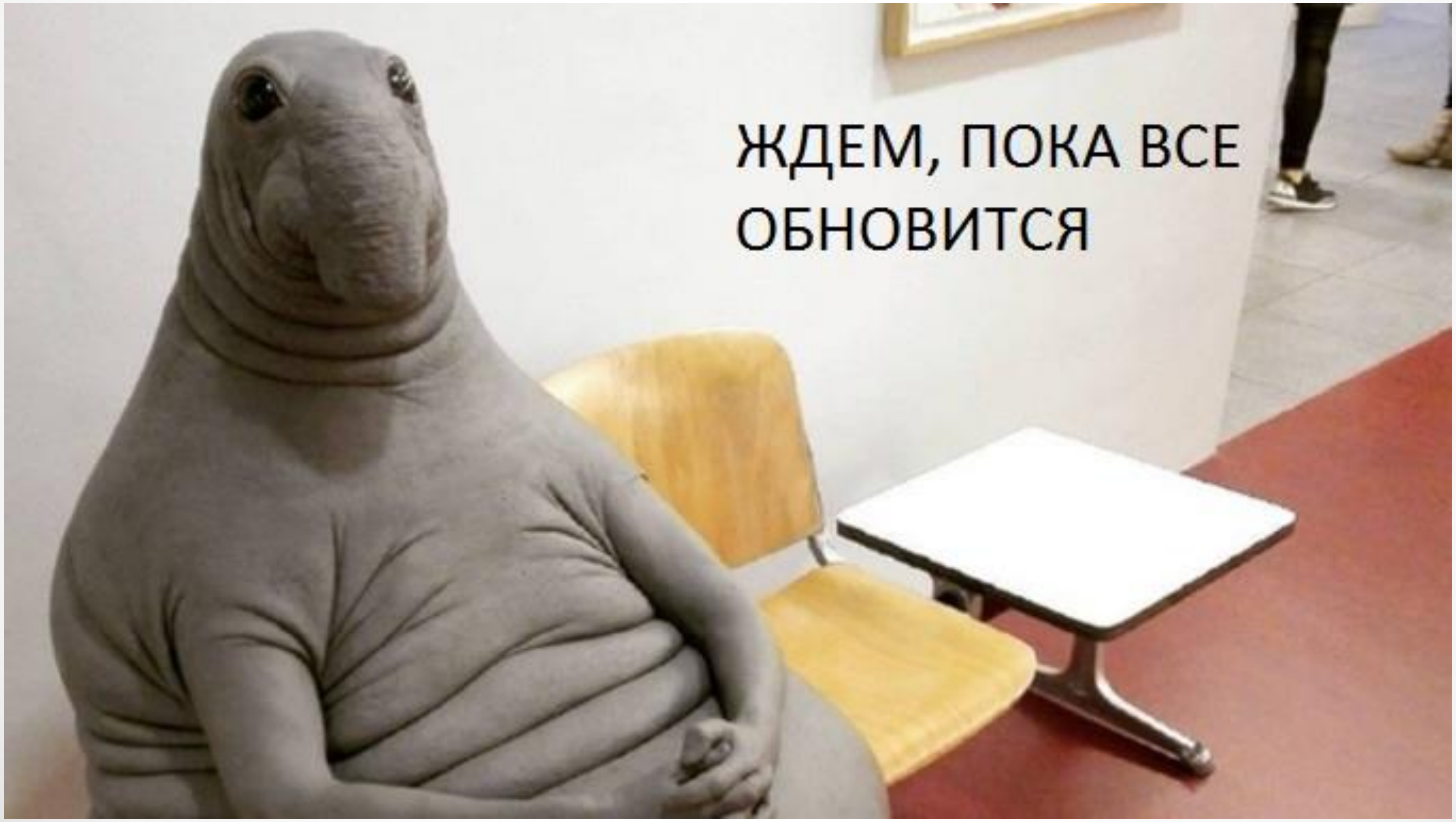


```
this.userForm.get('username').setAsyncValidators(
  this.validateUniqueName);
```

Task #7. Custom & async validators

1. Create your own validator function that checks that author field contains at least two words.
2. Add this function to the validators section of the **FormControl.control** method**.**
3. Check that submission and validation works.
4. Add new **asynchronous** validator that verifies if there already exists book with such title. Use book service to check book existence.

Change detection

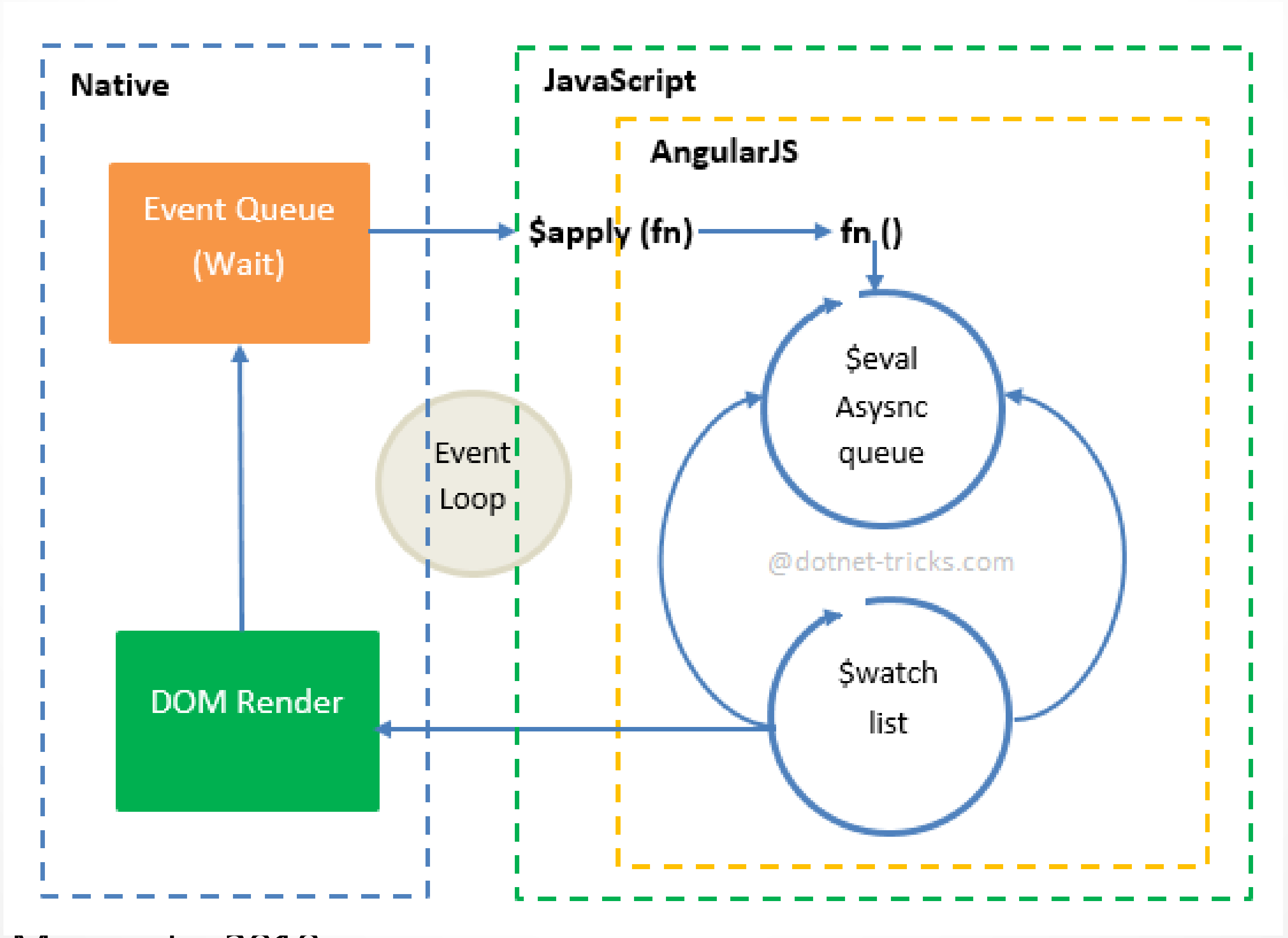


Change detection

Angular 1.Change detection

- ✓ Invoked by directives(**ng-model**), services**($http, $timeout**) or **$scope.$apply**()
- ✓ **Watcher** created for every dynamic expression
- ✓ After each event Angular triggers **digest cycle**
- ✓ During digest cycle every expression is evaluated and compared with old value
- ✓ This cycle is repeated until results are stable

Angular 1.Change detection



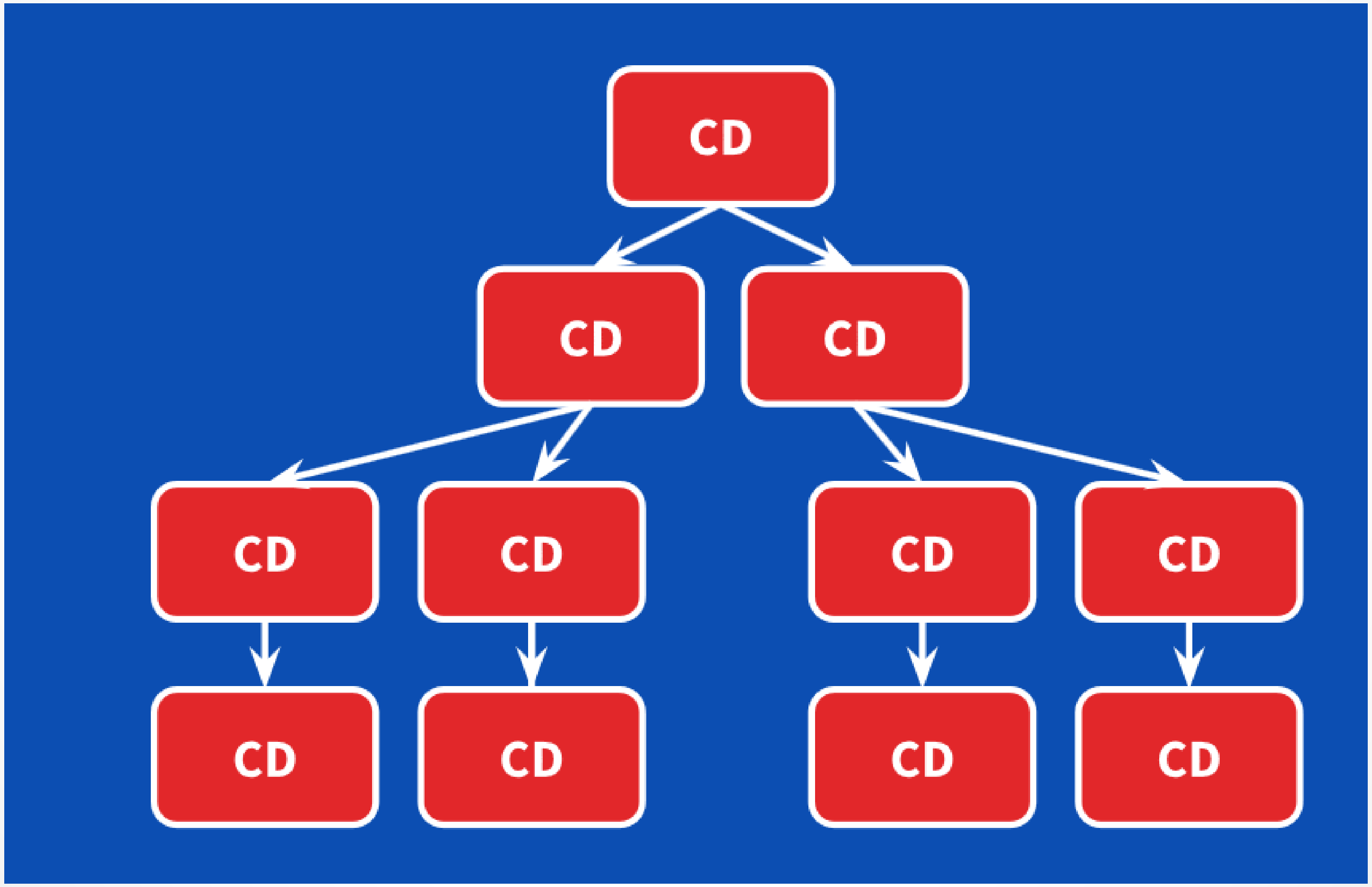
Change detection

- ✓ Component state can change by events, timers and other asynchronous tasks
- ✓ Each model change should reflect in DOM update
- ✓ Each component has its own change detector
- ✓ Angular uses **zones** to get notified about changes
- ✓ Zone is language feature in **Dart** ported to JavaScript as Zone.js
- ✓ A zone is an execution context that persists across asynchronous tasks(similar to **ThreadLocal** in Java)
- ✓ Zone.js patches all asynchronous runtimes and provides hooks(before, after, exception)

Change detection

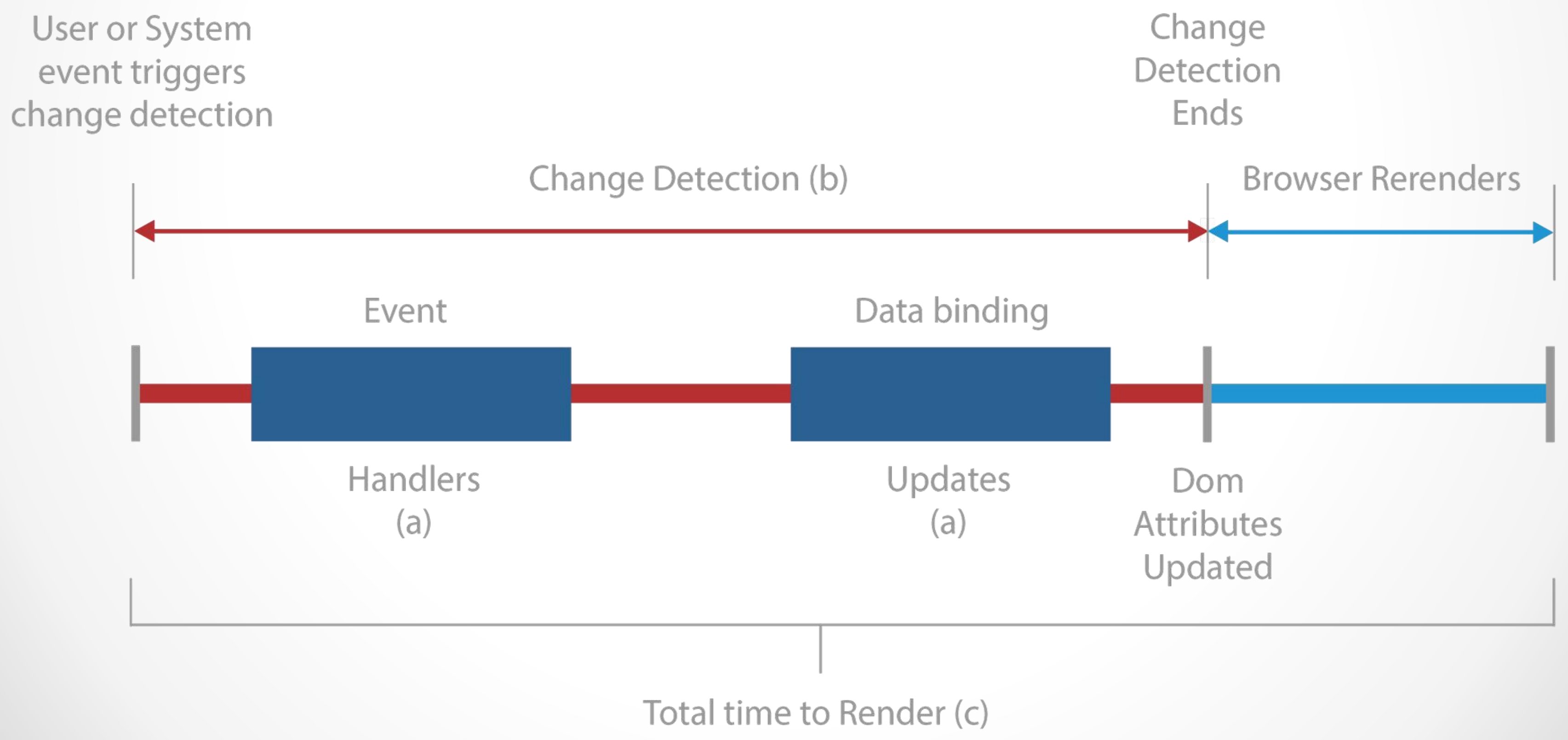
- ✓ Component can update state of its children but not ancestors
- ✓ If child component updates its parent then exception is thrown(in **development** mode)
- ✓ Single traversal across the component tree
- ✓ No more infinitive loops
- ✓ The whole process is faster

Change detection



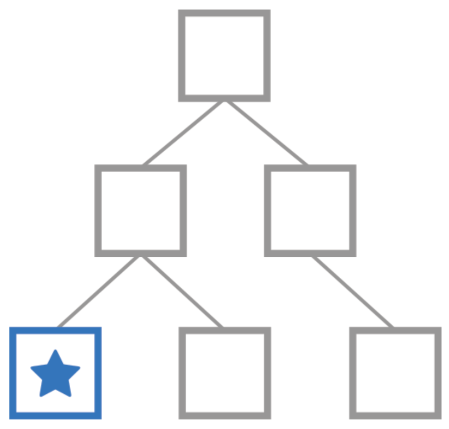
Runtime performance

- ✓ Total rendering time should be less than 17 ms to achieve 60FPS

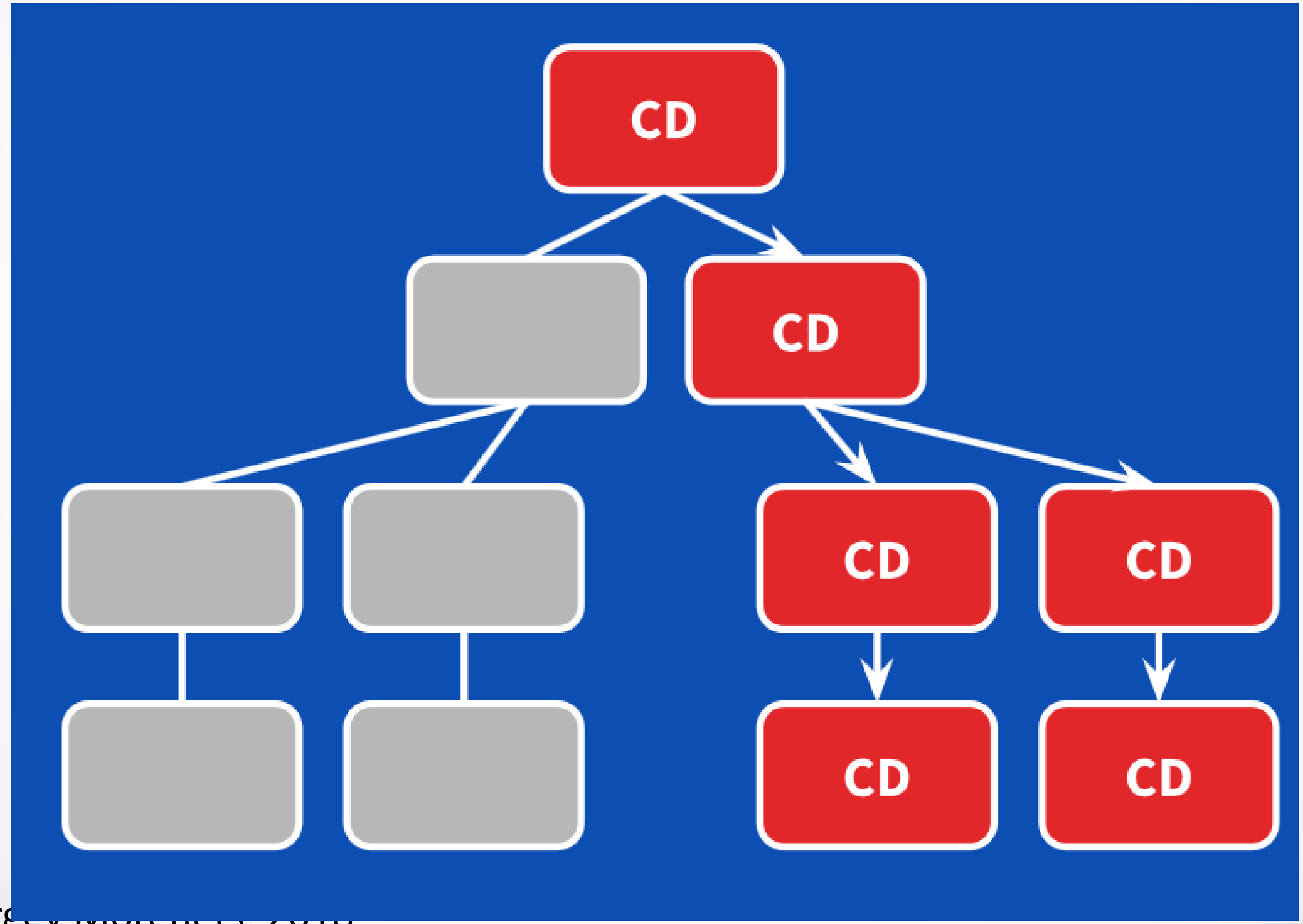


Runtime performance

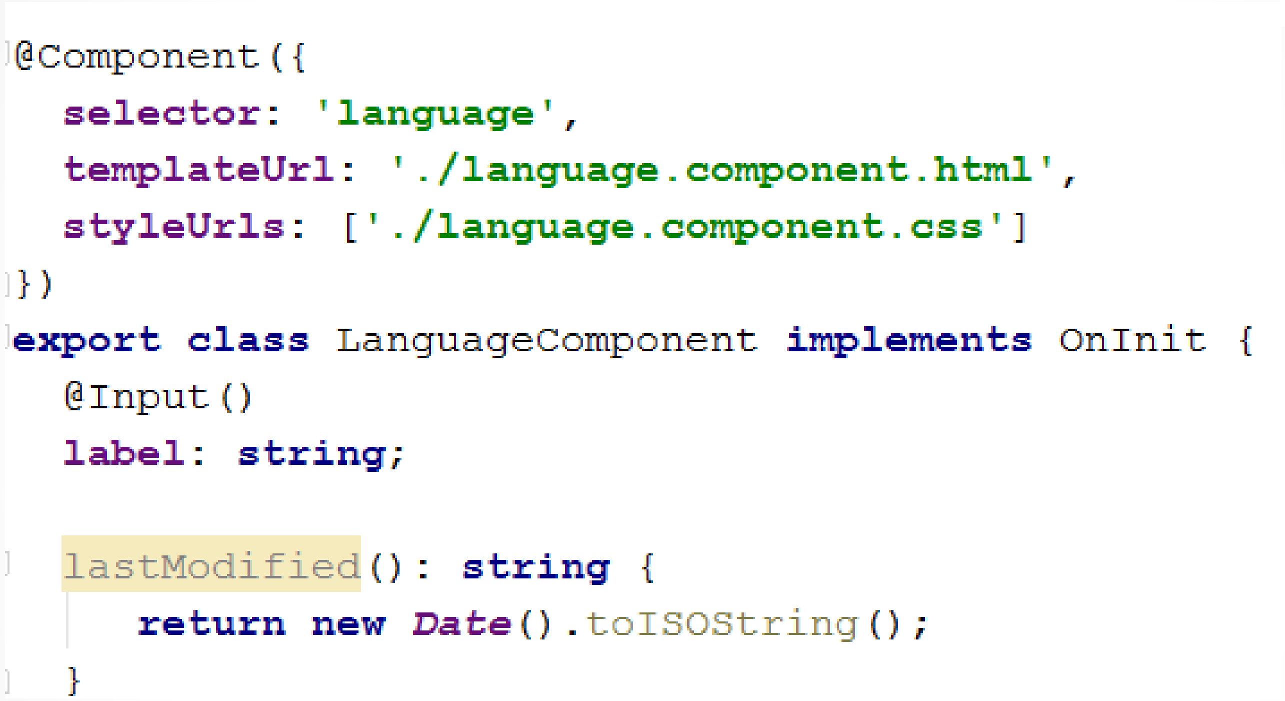
- ✓ Fast event handlers
- ✓ Reduce number of callback executions inside change detection cycle
- ✓ Reduce time of change detection cycle



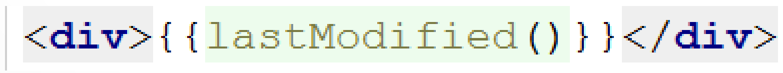
Change detection



Change detection

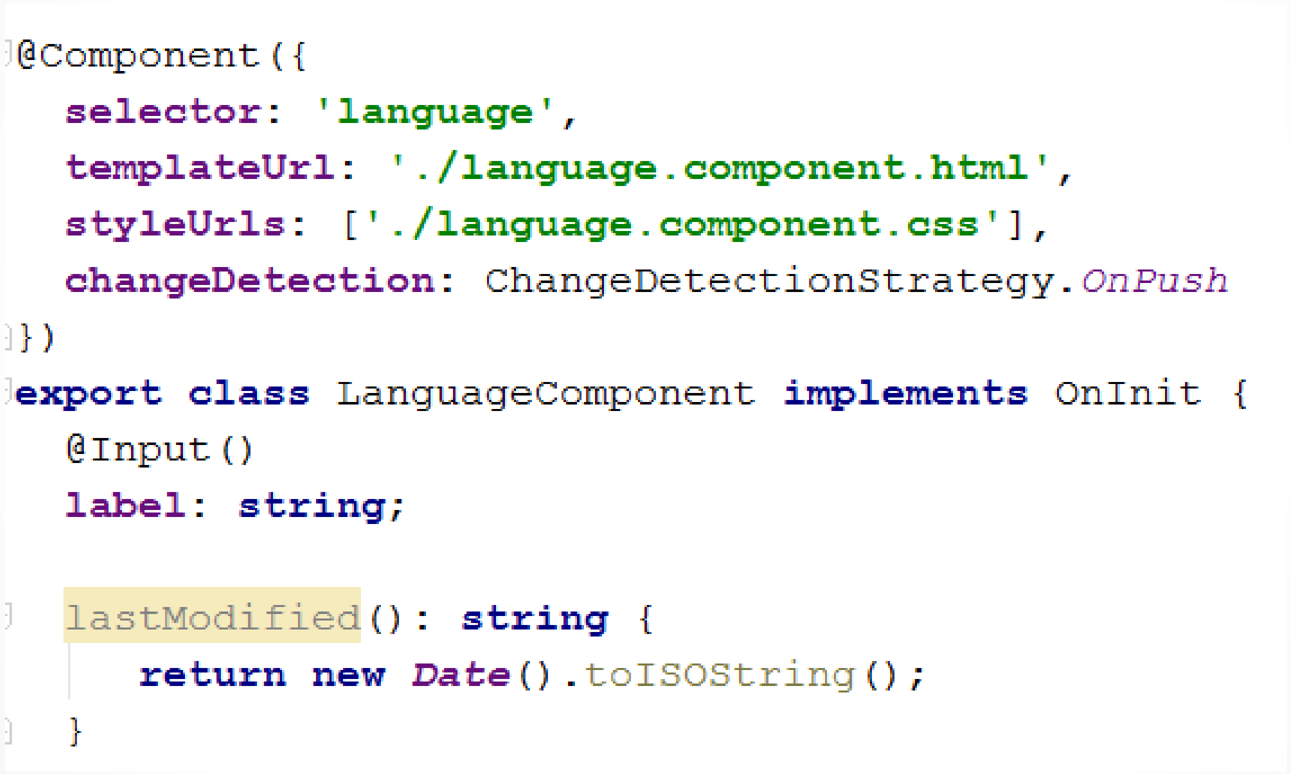


```
@Component({
  selector: 'language',
  templateUrl: './language.component.html',
  styleUrls: ['./language.component.css']
})
export class LanguageComponent implements OnInit {
  @Input()
  label: string;

  lastModified(): string {
     return new Date().toISOString();
  }
```



```
<div>{{lastModified()}}</div>
```

Change detection



```
@Component({
  selector: 'language',
  templateUrl: './language.component.html',
  styleUrls: ['./language.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class LanguageComponent implements OnInit {
  @Input()
  label: string;

  lastModified(): string {
    return new Date().toISOString();
  }
```

Task #8. Change detection

1. Create new component that will contain **@Input** property.
2. Add a method to new component that will increment and display a **counter**. Invoke it in the component template.
3. Add an event handler in the parent component that will change a model. Modify **changeDetection** attribute of **@Component** decorator in the child component
4. Run application and make sure it works properly.

Task #9. Global event bus. Sync version

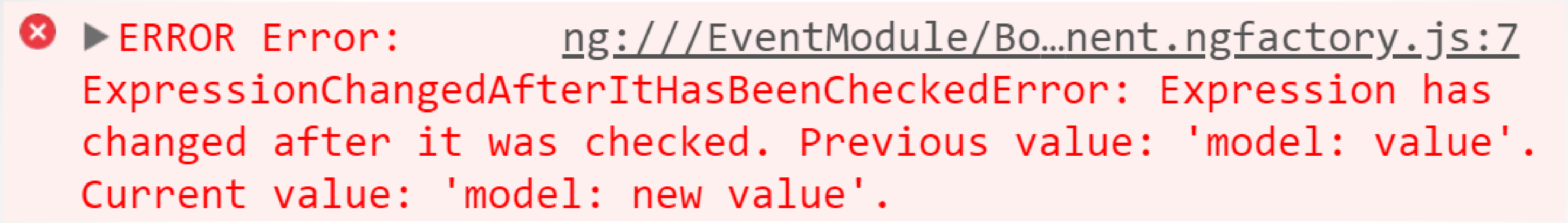
1. Create new interface **IEventConsumer** with single method
2. Create new interface **IEventBus** with subscribe/send methods. subscribe method should accept parameter of **IEventConsumer** type. Let AsyncEventBus class implement **IEventBus** interface.
3. Create new synchronous implementation of IEventBus that simply stores consumers in the **Array<IEventConsumer>**

Component sample

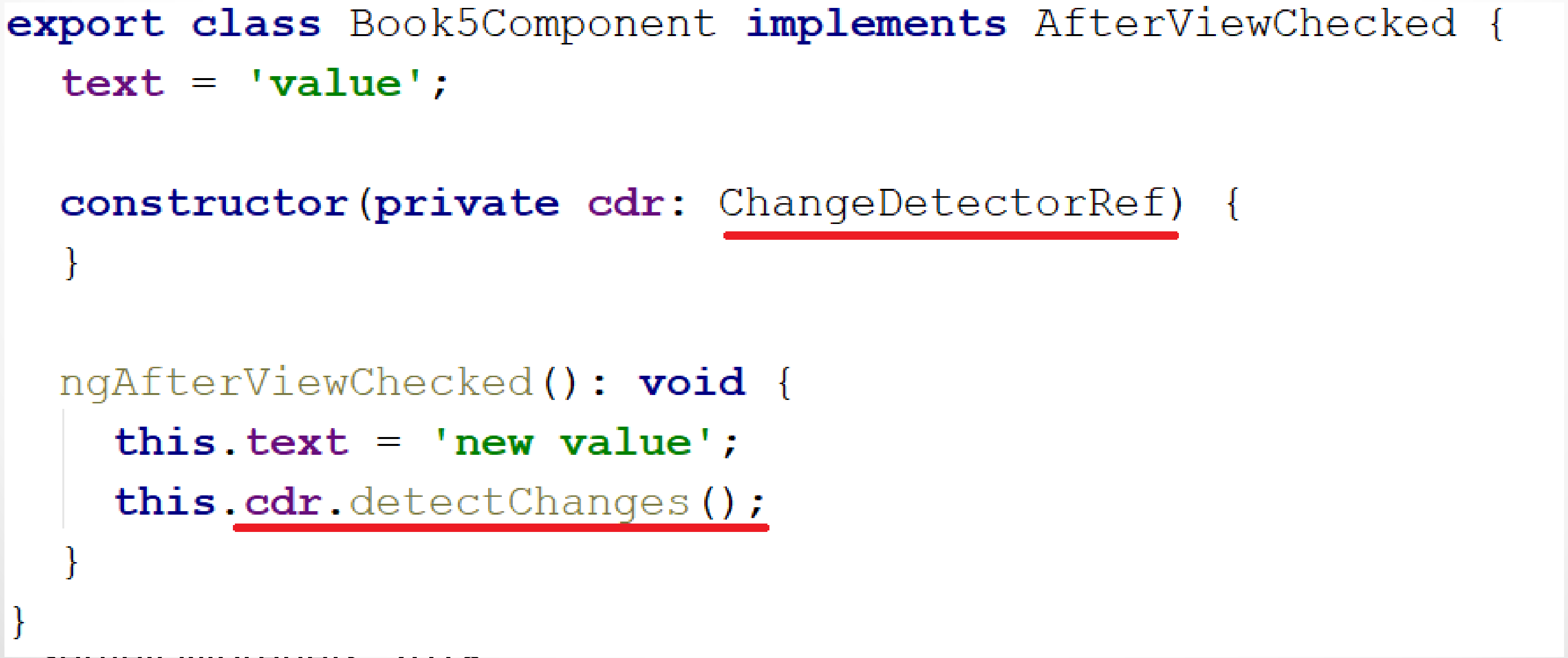
```
@Component({
  selector: 'app-book5',
  template: '<input [(ngModel)]="text"/>{{text}}'
})
export class Book5Component {
  text = 'value';
}
```

```
export class Book5Component implements AfterViewChecked {
  text = 'value';

  ngAfterViewChecked(): void {
    this.text = 'new value';
  }
}
```

Change detection

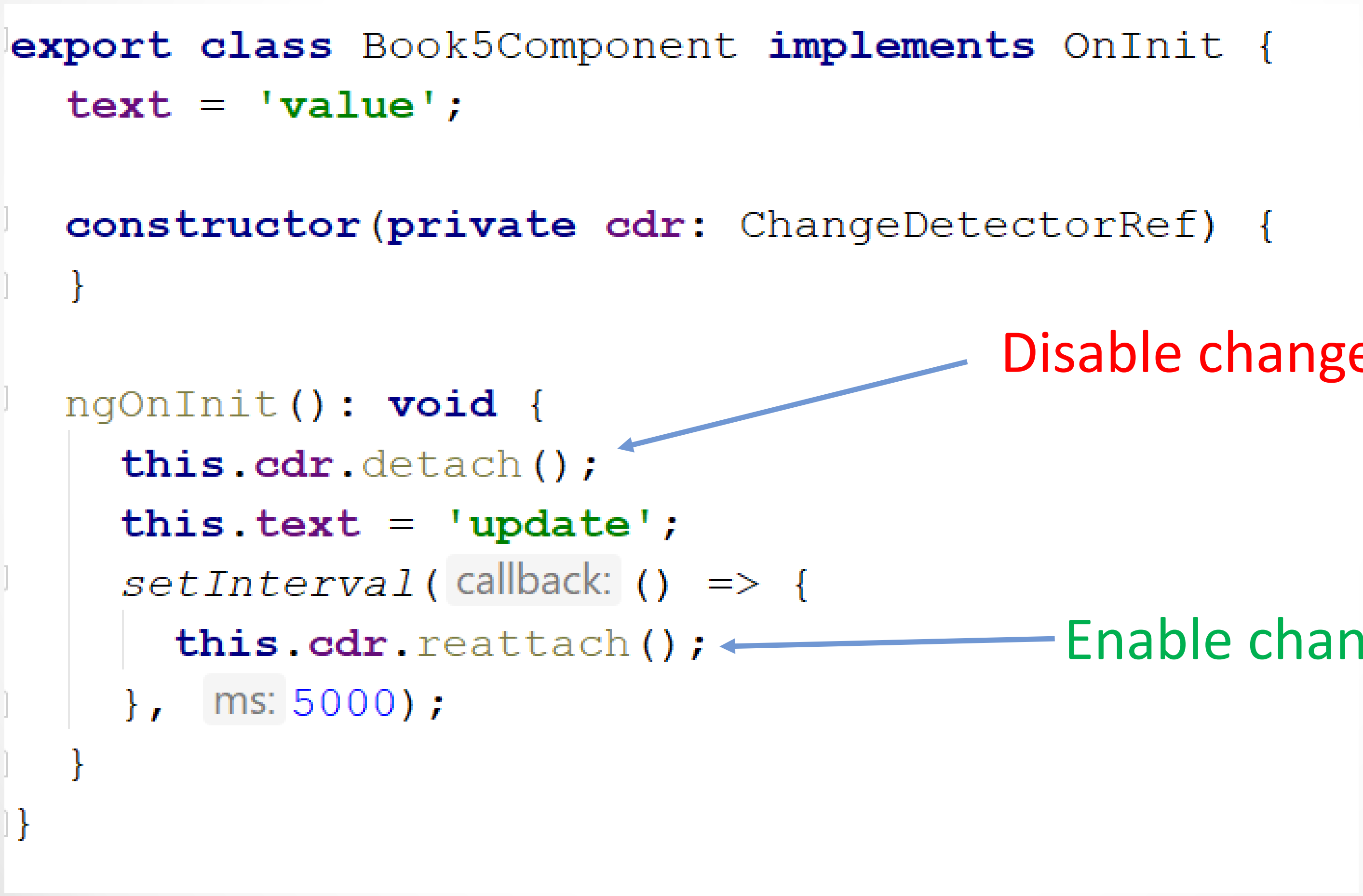


```
▶ERROR Error:        ng:///EventModule/Bo…nent.ngfactory.js:7
ExpressionChangedAfterItHasBeenCheckedError: Expression has
changed after it was checked. Previous value: 'model: value'.
Current value: 'model: new value'.
```



```
export class Book5Component implements AfterViewChecked {
  text = 'value';

  constructor(private cdr: ChangeDetectorRef) {
  }

  ngAfterViewChecked(): void {
    this.text = 'new value';
    this.cdr.detectChanges();
  }
}
```

Change detection



```
export class Book5Component implements OnInit {
  text = 'value';

  constructor(private cdr: ChangeDetectorRef) {
  }

  ngOnInit(): void {
    this.cdr.detach();
    this.text = 'update';
    setInterval( callback: () => {
      this.cdr.reattach();
    },  ms: 5000);
  }
}
```

Disable change detection

Enable change detection

Task #10. Programmatic change detection

1. Try to enable/disable component check detection using detach/reattach functions of **ChangeDetectorRef** object.
2. Update getBooks() function in **BookService** class so that it returns list of books in the random order.
3. Modify BooksComponent so that it refresh list of books each 3 seconds.

Component management

```
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styles: []
})
export class BookComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

```
<app-book></app-book>
```

Component management

```
@ViewChild('parent', {read: ViewContainerRef})
private parent: ViewContainerRef;

constructor(private componentFactoryResolver:
                 ComponentFactoryResolver) {
}

ngOnInit(): void {
  const bookComponent = this.componentFactoryResolver
    .resolveComponentFactory(BookComponent);
  this.parent.createComponent(bookComponent);
}
```

parent.component.ts

```
<ng-template #parent></ng-template>
```

parent.component.html

```
entryComponents: [BookComponent]
```

app.module.ts

Task #11. Component management

1. Create new component **BannerHeaderComponent**
2. Create new components BestBuyComponent and DiscountsComponent that will display ad content.
3. Create new service **BannerService** that should return random banner component and change it every 5 seconds.
4. Update BannerHeaderComponent to use BannerService and dynamically change banners provided by the service.

- ✓ Sergey Morenets, sergey.morenets@gmail.com