ReactiveX

Programowanie reaktywne w aplikacjach mobilnych

Czym jest programowanie reaktywne?

Programowanie reaktywne jest asynchronicznym paradygmatem opartym na sekwencjach danych oraz obserwatorach.

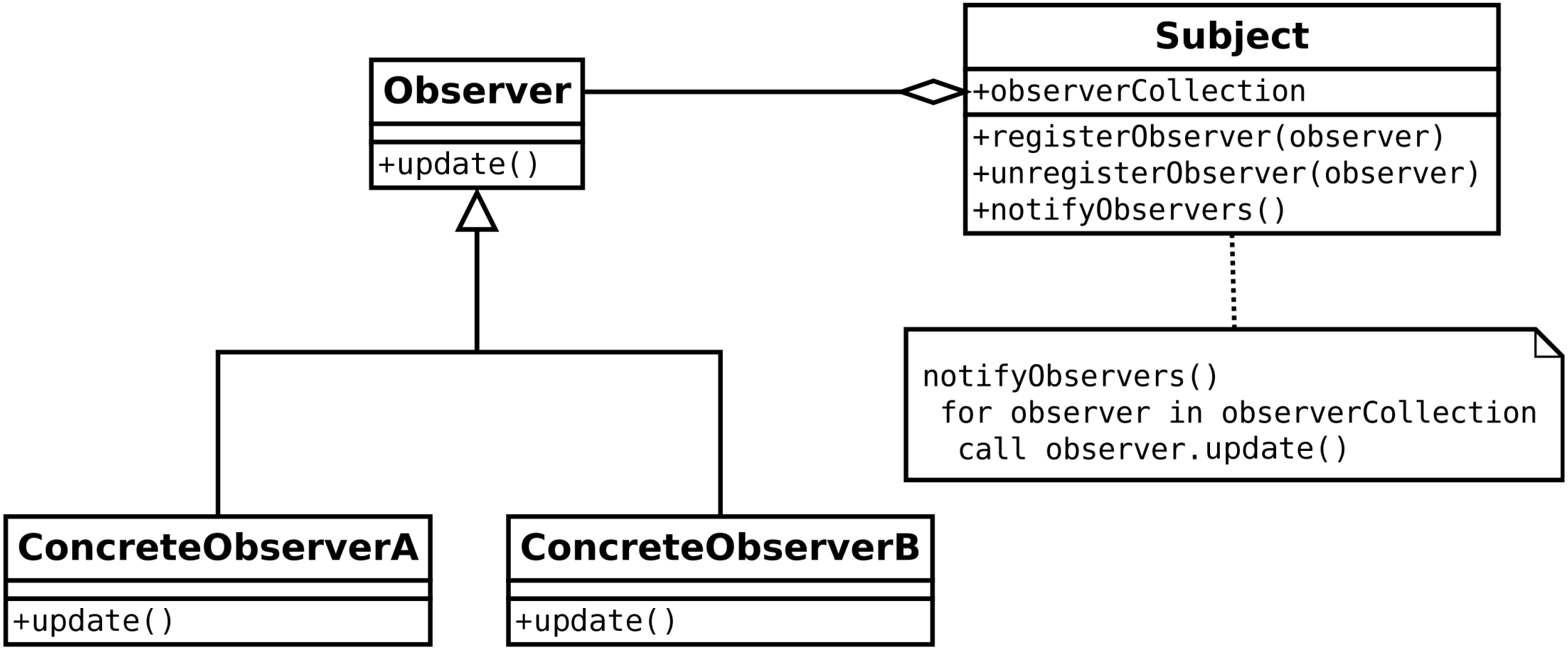
> Paradygmat
>
> 1. «przyjęty sposób widzenia rzeczywistości w danej dziedzinie, doktrynie itp.»
> 2. «zespół form fleksyjnych danego wyrazu lub końcówek właściwych danej grupie wyrazów»
>
> https://sjp.pwn.pl/slowniki/paradygmat.html

ReactiveX - meta-biblioteka?

ReactiveX to kolekcja otwartoźródłowych bibliotek w konkretnych językach, które implementują wzorce opisane przez dokumentacje ReactiveX.

RxJava, RxKotlin, RxSwift, RxDart, Rx .Net, RxJS...

Wzorzec obserwatora



Obserwujący jest dobrym uzupełnieniem w dostępie do asynchronicznych sekwencji w kolekcjach

	single items	multiple items
synchronous	`T getData()`	`Iterable<T> getData()`
asynchronous	`Future<T> getData()`	`Observable<T> getData()`

Co nam ułatwia ReactiveX?

Możesz wykonywać operacje na `Observable` w bardzo podobny sposób jak na listach, czy tablicach. Nie musisz mieć w głowie zarządzania wątkami czy synchronizacji.

event	Iterable (pull)	Iterable (pull)
retrieve data	`T next()`	`onNext(T)`
discover error	throws `Exception`	`onError(Exception)`
complete	`!hasNext()`	`onCompleted()`

High-order functions

Iterable

```
getDataFromLocalMemory() .skip(10) .take(5) .map({ s -> return s + "
transformed" }) .forEach({ println "next => " + it })
```

Observable

```
getDataFromNetwork() .skip(10) .take(5) .map({ s -> return s + "
transformed" }) .subscribe({ println "onNext => " + it })
```

Gang of Four

> Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku

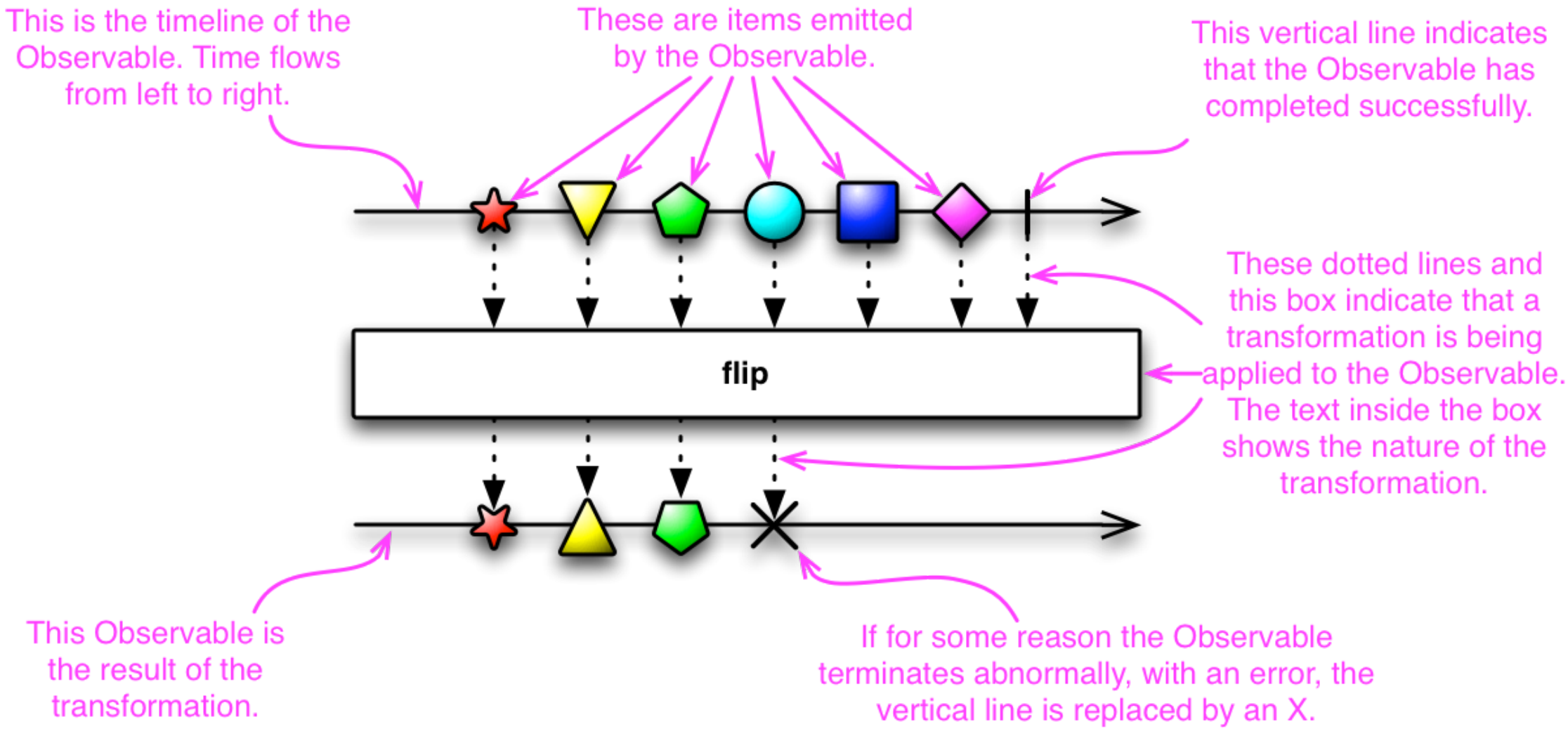
`Observable` w rozumieniu RX rozszerza wzorcowy wzorzec obserwatora o dwa brakujące elementy.

1. Sygnalizacja, kiedy nie ma więcej danych do przetworzenia - `onCompleted`
2. Sygnalizacja błędu - `onError`

Observable

W ReactiveX ṙ ű ẎǴw̃y l Ấ ẃẎẴú Ỵ̈ w̃ă ű ẴḻǴ Ẏệ’ do È ű ẎǴẉỴ ű ḵǴ.
Obserwator reaguje na to jakie zdarzenia ỹ ăǴṊ ệẪḻǴ Observable.
Dzięki takiej konstrukcji nie blokujemy wątku, tylko nasłuchujemy na zdarzenia i reagujemy dopiero jak coś przyjdzie.

Marble diagrams



Jak zacząć - standardowe wywołanie

1. Wywołaj metodę.
2. Przypisz zwróconą wartość do zmiennej.
3. Użyj tej zmiennej.

```
int students = getNumberOfStudents()
```

```
students ...
```

Jak zacząć - asynchroniczne wywołanie

1. Zdefiniuj metodę, która robi coś jak przyjdą dane (np. odśwież widok). Ona nie zostanie wywołana. To jest część obserwatora.
2. Zdefiniuj metodę, którą chcesz wywołać (np. pobranie listy studentów z internetu). To jest Observable.
3. Zasubskrybuj obserwatora do Observable. To już wywoła pobranie listy studentów.
   `observable.subscribe(observer)`
4. Jeśli Observable coś zwróci, metoda do która coś robi np. na UI zostanie wywołana.

```
def refreshUI = updateLabels()
def getStudentsObservable = getStudentsFromNetwork()
getStudentsObservable .observe(refreshUI)
...
```

onNext, onCompleted, and onError

```
def myOnNext = { item -> /* do something useful with item */ };
def myError = { throwable -> /* react sensibly to a failed call */ };
def myComplete = { /* clean up after the final response */ };
def myObservable = someMethod(itsParameters);
myObservable.subscribe(myOnNext, myError, myComplete);
// go on about my business
```

Composition via Observable Operators

Observable i Obserwator to tylko implementacja wzorca projektowego. Prawdziwą siłą ReactiveX są v́Ǵ ùÂẸỲǴǴŽĂǴÕỲẹ ÕẎ.
Pozwalają one łączyć, przekształcać dane emitowane przez Observable w sposób asynchroniczny.

Większość operatorów zwraca Observable, więc można je łączyć jeden po drugim - chain.

Zobaczmy to w akcji

http://reactivex.io/documentation/operators.html