**Reactive programming**

**What is Reactive Programming?**

Reactive Programming is a programming language with asynchronous data stream .Once an event will raise it will react with responsive and non-blocking manner that’s why it named it as reactive programming.

In a Reactive Programming context, “Everything is a Stream and acts in a non-blocking manner when there is data in the stream.”

This Reactive programming in java is introduced by Netflix organization with API (Rx-Java)

**Why we need Reactive Programming?**

When we have traditional way to develop application why we should go for reactive programming this should be the common question for all

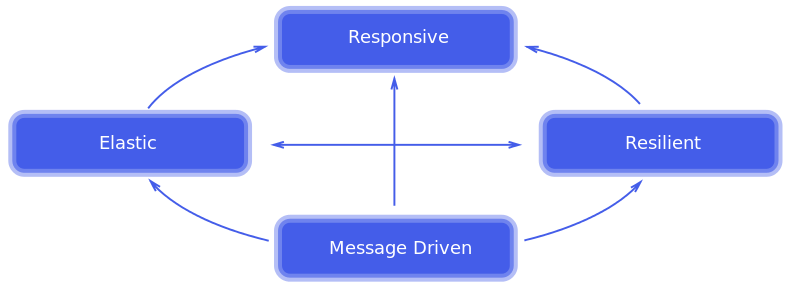
There is four pillar to move towards Reactive Programming

1. Responsive

2. Resilient

3. Elastic

4. Message Driven



**Responsive:**

Responsive means if we are raising any events on stream it will return response in fraction of time as message processing in highly concurrent environments

**Resilient:**

Resilient means Application should be responsive at the time of failure

Normally we are integrating multiple modules and each are depend to each other assume one module is failing so it should not be impact all, it should be propagate

**Elastic**:

Elastic means our system should be handle N number of request, it should be well capability to load balance at any condition

**Message Driven:**

Message Driven means asynchronous flow of execution, where we no need to wait for response after send request to server , once we send request it move to next business it shouldn’t depends on first one response , the first request response should be handle by callback mechanism

**Ideal Use Cases for Implementation of Reactive Programming**

1. A large number of transaction-processing services, as in the banking sector.
2. Notification services of large online shopping applications, like Amazon.
3. A share trading business where share prices change simultaneously.

**What is Rx-Java?**

RxJava is a Java VM implementation of ReactiveX (Reactive Extensions): a library for composing asynchronous and event-based programs by using [observable](http://www.vogella.com/tutorials/DesignPatternObserver/article.html) DP.

RxJava tries to be very lightweight. It is implemented as a single JAR that is focused on just the Observable abstraction and related higher-order functions

In simple word Rx-Java is an API which support us for reactive programming

Observer Design principle:

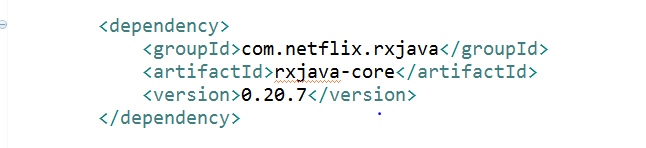
The observer pattern defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.

The object which is being watched is called the subject. The objects which are watching the state changes are called observers or listeners

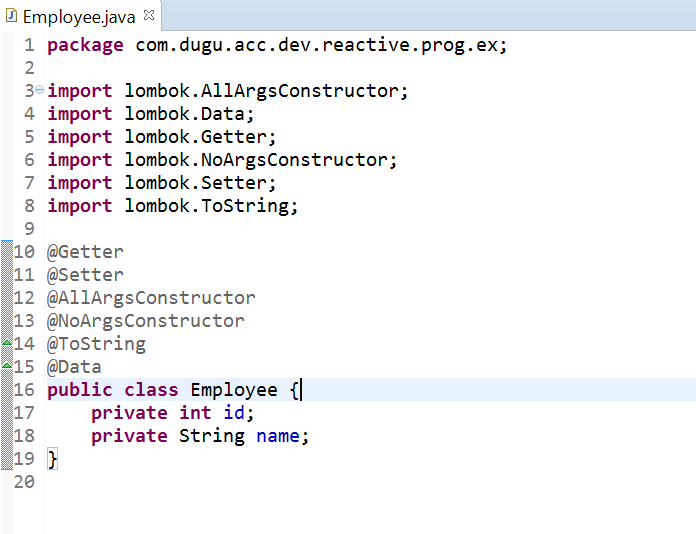
For more visit: [observable](http://www.vogella.com/tutorials/DesignPatternObserver/article.html)

Let’s do a small POC so that we will get to know about the design principle

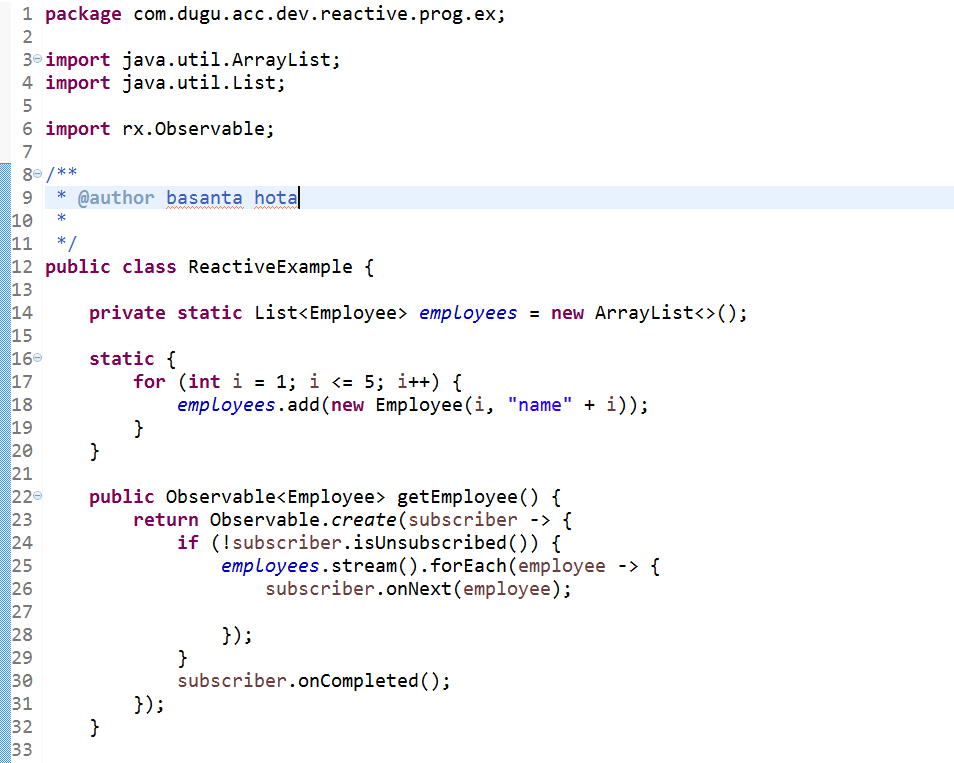
First create one standalone maven project then add below dependency to achieve Reactive programming in java

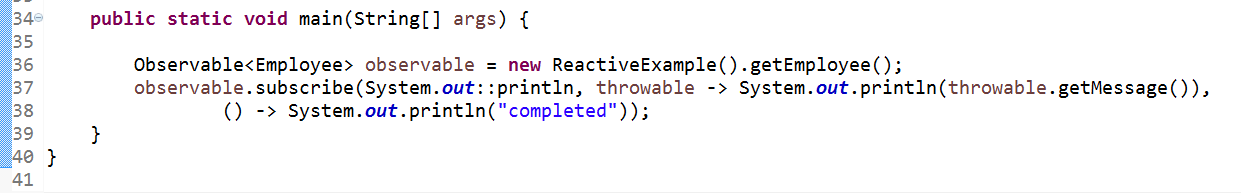


Entity class:



Create a class and write business with Observable so that it will notify to other once task is done like below





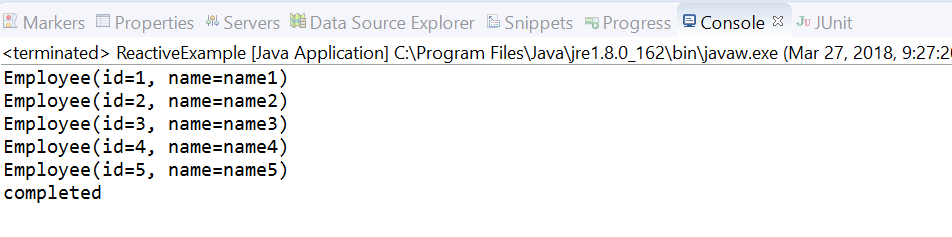
Now let’s understand step by step

Initially we are invoking getEmployee () method so control goes to it

Form getEmployee () method we are creating Observable stream and checking each subscriber is subscribe or not

If subscribes then we are iterating that stream and we are calling subscriber.onNext (Object obj) so that it will return to call back once each subscriber verified and execute logic at end of loop we are calling subscriber.onCompleted() so that on completion of task it will notify to all subscriber

Response of above code



Line number 37 indicating subscribe method arguments, subscribe () is an overloaded method



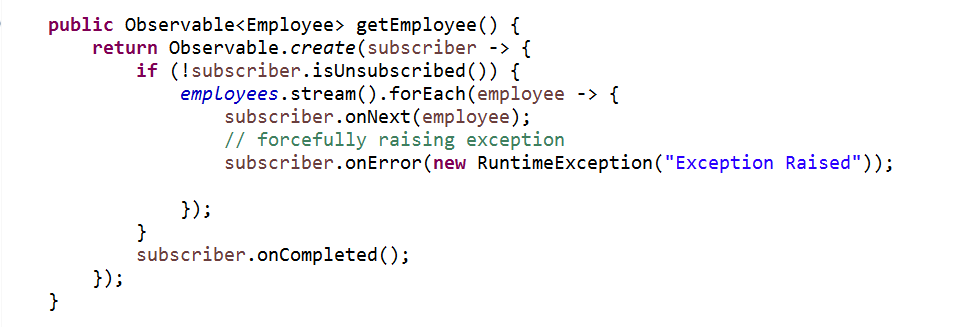
As per method signature

First argument will take onNext() call which is callback

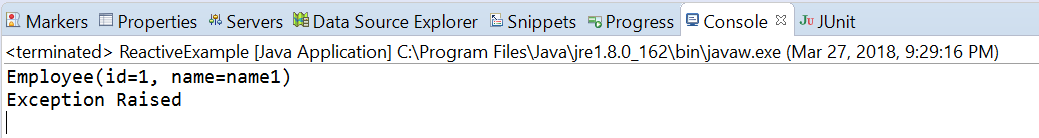
Second argument will take care when we have an exception

Third argument will take care about on complete action

As per above code we don’t have any exception so second argument value not excuted now let’s create an exception manually to check the complete flow of callback excution when a subscriber subscribe an event



Response:



This is all about the observable principle followed by Rx-Java to support Reactive programming

**Source code:** [**Download**](https://github.com/Java-Gyan-Mantra/reactive-programming)