Angular Js RxJs

Agenda

- Asynchronous Programming
- Promises
- Observables
- Observer Pattern
- Useful Resources
- Data Stream

Asynchronous Programming

- Not occurring at the same time.
- Code executes one line at same time- sometimes before first function has completed.
- It was fast but tends to render empty pages before server responses.
- Call back solved this problem for a while- wait for this code to complete before moving forward

- Function myfunction(options, callback){
- Client.get('http://localhost?args' + options,function(response){
- callback(response)});
- }

Promises

- But did not work in complex application- nested callback problem, difficult to error handling.
- Promises came into picture
- Easy error handling, chaining promises
- Promise is some placeholder which return promise to let us know that we were waiting on something

- **Var promise= new Promise((resolve,reject) =>{**
- **Business logic, if we get some data resolve else reject(Error('error'));**
- **}**
- **});**

- Promises also gives a function then which is a trigger to tell us promise is completed.
- Promise.then((result))=>{
- });

- Real time updates, Analytics and data should be available various devices.
- We can only use each promise once.

Observables

- **Basically reusable promise that keeps listening after the “then” method**
- **.subscribe is similar to .then except it is reusable.**

- **Var observable= Rx.Observable.create((observer) =>{**
- **Observer.next(1);**
- **Observer.next(2);**
- **Observer.complete();**
- **});**
- **Observale.subscribe(**
- **Value => console.log(value);**
- **Err => console.log(err);**
- **() => console.log(“this is done”);**
- **);**

- Instead of then observable will have subscribe method to listen each update and respond.

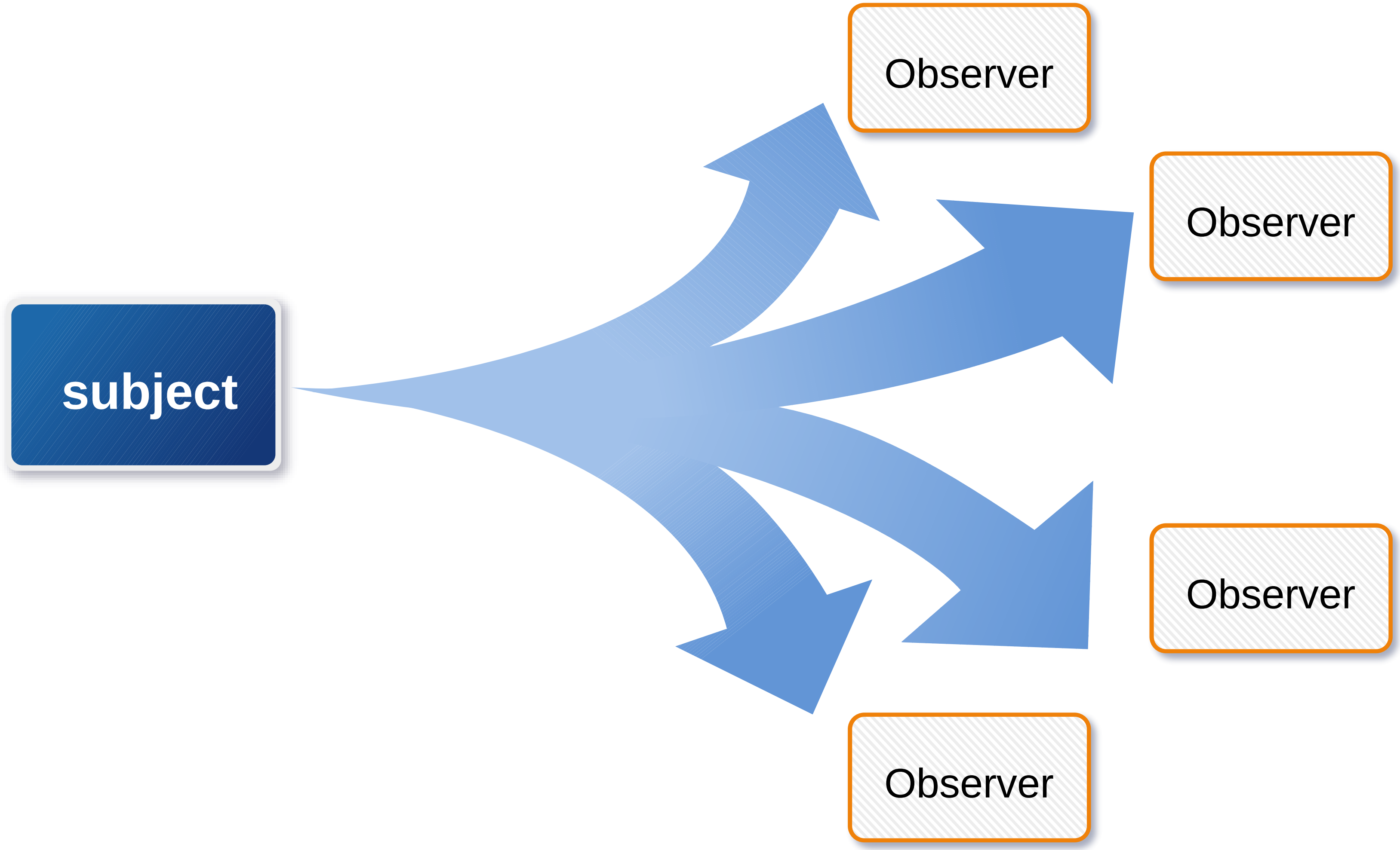
What is design pattern?

- **If code is not organized its messy**
- **Design and architecture of application**
  - Book- Gang of four design patterns

Observer Pattern

- Object(here subject) maintains a list of its dependents called observer and notifies them automatically when its state changes.
- Also known as event driven design.
- When subject gets updated all observers are notified with updated data.

Event Driven Design



Useful Resources: Reactive X

- **http://reactivex.io/**
- **https://rxjs-dev.firebaseapp.com/**

Database Observable

- **Its not just querying data any more.**
- **We want to know changes**
- **Data Stream- sequence of data elements made available over time- conveyor belt**
- **Payload- event update carrying new data**

Resources

- https://angular.io/
- https://rxjs-dev.firebaseapp.com/
- http://reactivex.io/