Modern apps

High data scale

High usage scale

Clous based costs

Scaling

* Vertical scaling – increase power of hardware
* Horizontal scaling -

**Future API**

* Future and CompleteableFuture are concurrency API’s

CompletableFuture<User> userAsync = CompletableFuture.supplyAsync(()->

userService.getUser(userId));

CompletableFuture<User> userPreferencesAsync = CompletableFuture.supplyAsync(()->

userPreferenceService.getPreferences();

CompletableFuture<User> bothFutures = CompletableFuture.allOf(userAsync, userPreferencesAsync);

bothFutures.join();

Text

Description automatically generated

Join is blocking in above code-base

**Reactive-Programming**

Text

Description automatically generated

* If RHS of map is value, then use map
* If RHS of map is a stream, then use flat map

UserStream.map(user->user.getFirstName());

UserStream.flatMap((id->Streamsources.userStream().filter(user->user.getId()==

* Patterns are 2 types

1. Iterator
2. Subscriber

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generatedDiagram

Description automatically generated

Diagram

Description automatically generated

Iterator pattern – myList.forEach(element->system.out.println(element));

Observer pattern – clicksChannel.addObserver(event-> system.out.println(element));

Integer value = ReactiveSources.intNumberMono().block();

In reactive world, subscribe is similar to forEach

Diagram

Description automatically generated

Flux/Mono is the publisher and we do

ReactiveSources.*intNumbersFlux*().subscribe(e-> System.*out*.println(e));

i.e publisher.subscribe(e-> System.*out*.println(e));

* Mono is asynchronous sequence of zero/one item
* Flux is asynchronous sequence of zero/n items
* Incase of flux, sometimes subscriber may not be able to process all the messages (because of speed) received, which may lead to backpressure
* At any point mono/flux can return 3 types i.e

1) Item

2) on complete

3) On error

2 and 3 are terminal events

Text

Description automatically generated

* ReactiveSources.*intNumbersFlux*().  
   subscribe(  
   number-> System.*out*.println(number),  
   err-> System.*out*.println(err.getMessage()),  
   ()-> System.*out*.println("complete"));

Base-subscriber has a class that can replace all the above 3 lambdas and get the work done

* Blocking to get value

Integer value = ReactiveSources.intNumberMono().block();

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

* .log() always needs subscribe to print the elements

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated