WELCOME
&
THANK YOU

<Creative Devs. Software/>

Reactive & Event Driven Programming

Level 1

MEET YOUR BITTY BYTE TEAM

TANGY F.

CEO

EDDIE K.

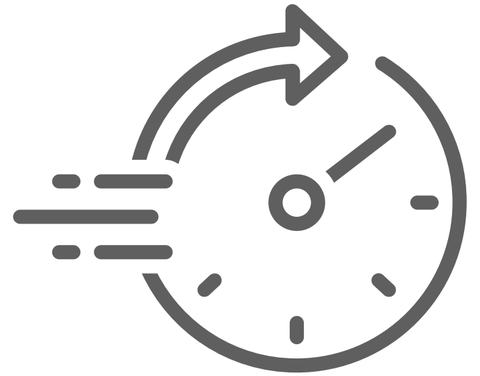
DEVELOPER

WILLIAM D.

DEVELOPER

Rapid Review

TRIVIA



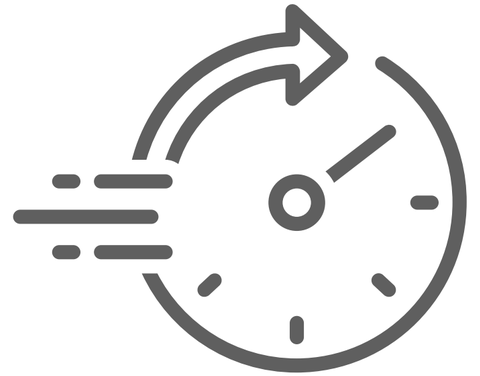
Rapid Trivia

What is the Turing Test and who was its author

Rapid Review

TRIVIA

Alan Turing 1912 - 1954



The Turing Test is a test of a machine's ability to exhibit intelligent behavior. The test, originally called the Imitation Game, was conceived by British mathematician Alan Turing

CODING EXERCISE

Time to try it yourself. Write the following class and run it. It will produce an error. To correct, add the appropriate backpressure strategy that will produce numbers indefinitely from 0 to n. You will need to stop the execution manually.

```java
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;
import java.time.Duration;

public class Practice4a {
    Run | Debug
    public static void main(String[] args) throws InterruptedException {

        Flux.interval(Duration.ofMillis(millis:1))
            .concatMap(a -> Mono.delay(Duration.ofMillis(millis:100)).thenReturn(a))
            .doOnNext(item -> System.out.println("Sending " + item))
            .blockLast();
    }
}
```

Let's see the code.

TODAY'S AGENDA

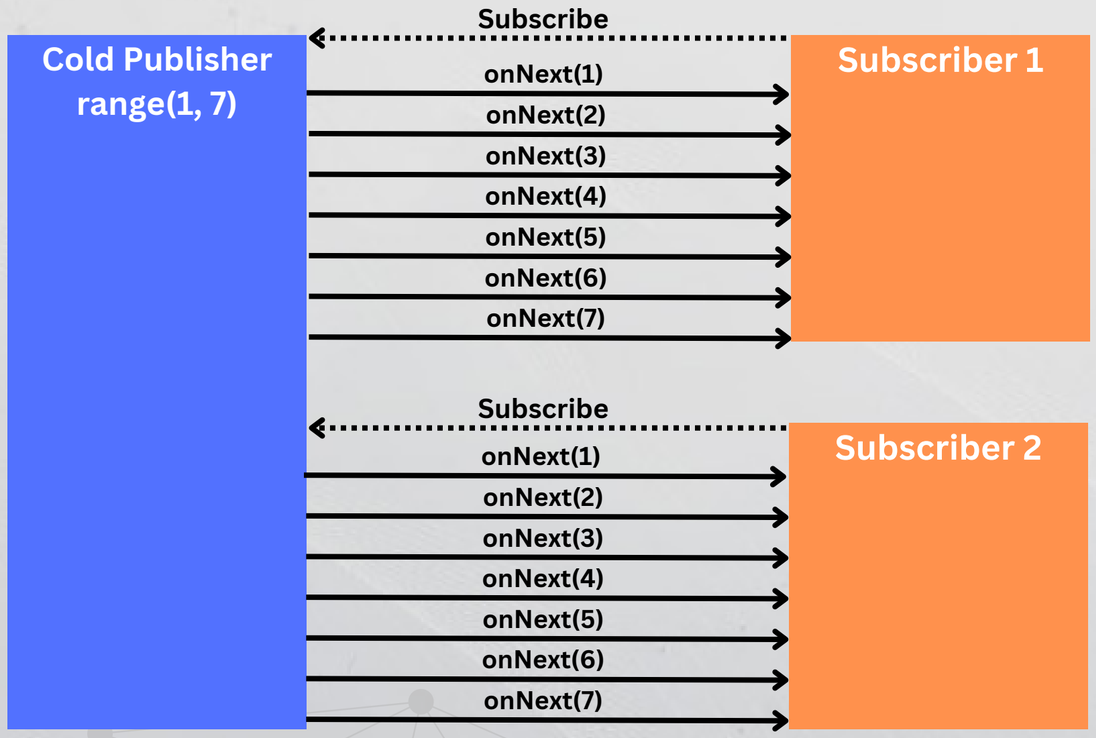
DAY 5

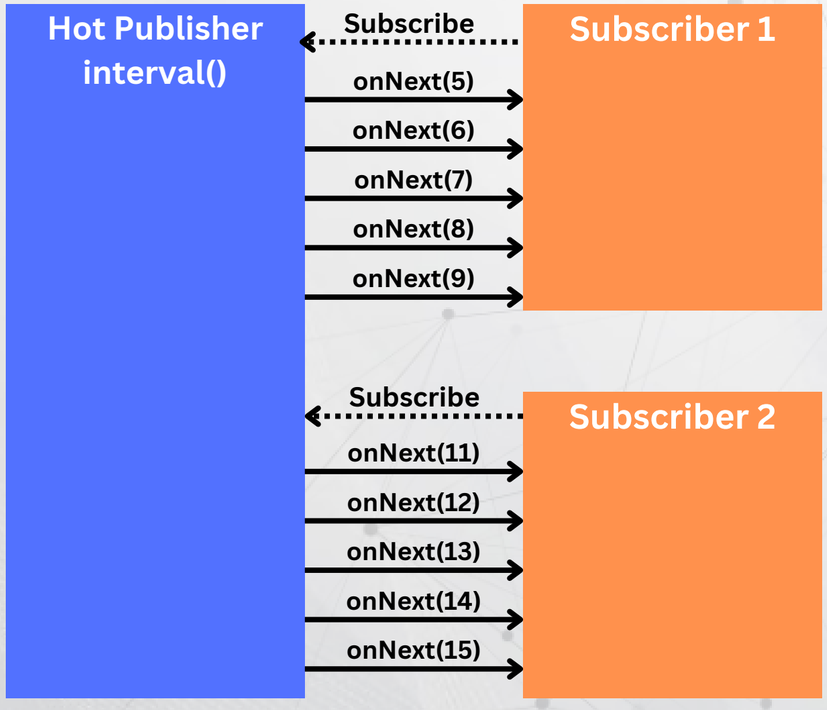
1. Reactive Hot and Cold Streams
2. Reactive Hot and Cold Streams with Kafka

DAY 5
LESSON 1

Reactive Hot and Cold Streams

HOT & COLD STREAMS





HOT STREAMS

Create a **ConnectableFlux**

Instantiate a Flux, then use delayElements().publish().connect() to start streaming

```
Flux<Customer> hotStream = customerController.getAllCustomers();
ConnectableFlux<Customer> connectable = hotStream
        .delayElements(Duration.ofMillis(millis:10))
        .publish();
connectable.connect();
```

Subscribe to start receiving the stream.

```
connectable.subscribe(data -> System.out.println("Subscriber 1: "+data));
```

Let's see the code.

DAY 5
LESSON 2

Reactive Hot and Cold Streams with Kafka

PRODUCE STREAMS WITH KAFKA

Using **KafkaTemplate** to produce

Instantiate a KafkaTemplate object.

```
@Autowired
private KafkaTemplate<String, Greeting> greetingKafkaTemplate;
```

Execute KafkaTemplate.send(). Pass topic name and payload as parameter.

```
greetingKafkaTemplate.send(greetingTopicName, greetingMessage);
```

Let's see the code.

HOT STREAM SUBSCRIBE WITH KAFKA

Using **KafkaConsumer** to receive

Instantiate KafkaConsumer with appropriate properties;

- **include group id instead of individual,**
- **enable.auto.commit to true (default)**

then subscribe to the correct topic name.

```
consumerProperties.put(key:"group.id", greetingTopicName +"hot");
kafkaConsumer = new KafkaConsumer<>(consumerProperties);
kafkaConsumer.subscribe(Arrays.asList(greetingTopicName));
```

Loop to execute KafkaConsumer.poll() and loop on the return to process data received

```
while(true) {
    ConsumerRecords<String, Greeting> records = kafkaConsumer.poll(Duration.ofMillis(millis:100));
    for (ConsumerRecord<String, Greeting> record : records) {
        Greeting message = record.value();
        LOGGER.info(format:"KafkaColdStreamConsumer :: Received message: {}, offset = {} key ={} serializedSize = {}",
            message,record.offset(),record.key(),record.serializedValueSize());
    }
}
```

Let's see the code.

COLD STREAM SUBSCRIBE WITH KAFKA

Using **KafkaConsumer** to receive

Instantiate KafkaConsumer with appropriate properties;

- **include an individual group id,**
- **enable.auto.commit to false**
- **auto.offset.reset to “earliest”**

then subscribe to the correct topic name.

```
consumerProperties.put(key:"group.id", greetingTopicName + UUID.randomUUID().toString());
consumerProperties.put(key:"enable.auto.commit", value:"false");
consumerProperties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, value:"earliest");
kafkaConsumer = new KafkaConsumer<>(consumerProperties);
kafkaConsumer.subscribe(Arrays.asList(greetingTopicName));
```

Loop to execute KafkaConsumer.poll() and loop on the return to process data received

```
while(true) {
    ConsumerRecords<String, Greeting> records = kafkaConsumer.poll(Duration.ofMillis(millis:100));
    for (ConsumerRecord<String, Greeting> record : records) {
        Greeting message = record.value();
        LOGGER.info(format:"KafkaColdStreamConsumer :: Received message: {}, offset = {} key ={} serializedSize = {}",
            message,record.offset(),record.key(),record.serializedValueSize());
    }
}
```

LESSON QUESTION

Describe and/or give samples for :

- Asynchronous events
- Asynchronous Execution
- Asynchronous Streams

LESSON ANSWER

- Asynchronous events
  - Events emitted where the publisher does not wait for the subscriber to finish processing. Example; a mouse click where the user do not need to wait for the result, like in the submission of a form.
- Asynchronous Execution
  - When two or more process runs concurrently and non blocking instead of one after the other. Example; One process calculating interest earned for a costumers and the other calculate capital gains on investments for same costumers.
- Asynchronous Streams
  - When a stream of data is being sent to the subscriber and the publisher does not wait for each row to be processed by the subscriber before sending more. Example; Processing rows from a database query.

RESOURCES

Reactor Reference Guide

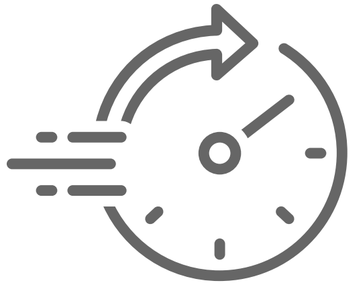
- https://projectreactor.io/docs

The Reactive Manifesto

- https://www.reactivemanifesto.org/
  - Click on the **Reactive Principles** link at the bottom of the page

Your IDE context sensitive help

POST-ASSESSMENT



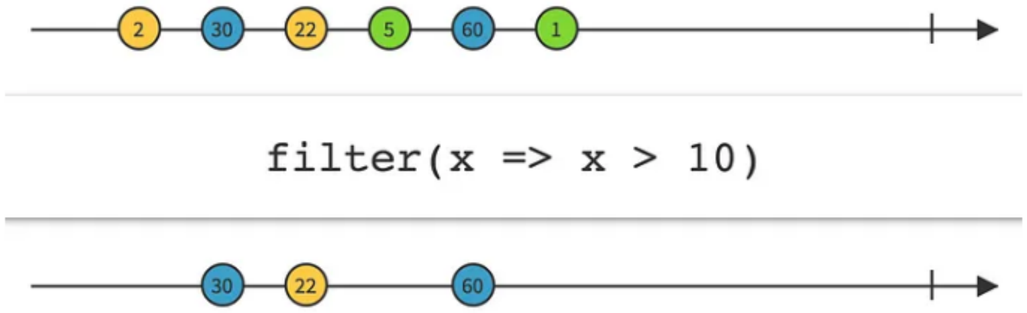
Event Driven * Reactive

On a blank document
Type your name
&
answers the questions
Add your doc.
to the shared drive

QUESTIONS 1 & 2 PRE- ASSESSMENT

Full Name: _____

1. What is the name given to to the diagrams shown bellow?



_____

2.In the code shown below, in which statement does the producer will start sending the values 1, 2, 3, 4, 5 to its consumer? Provide line number or statement.

```
2   Iterable<Integer> collection = List.of(1, 2, 3, 4, 5);
3
4   Flux<Integer> flux = Flux.fromIterable(collection);
5
6   flux.subscribe(
7           item -> System.out.println("Received: " + item),
8           error -> System.err.println("Error: " + error),
9           () -> System.out.println("Processing completed")
10  );
```

QUESTIONS 3 & 4 PRE- ASSESSMENT

3. What does the filter() operator do in reactive programming?

4. What is the difference between a factory operator and regular operator?

QUESTIONS 5 & 6 PRE- ASSESSMENT

5. What is backpressure within the context of reactive programming.

6. Describe what the following line of code will do.

```
Flux.range(start:1, count:10).subscribe(System.out::println);
```

HEARTFELT GRATITUDE
FOR YOUR PARTICIPATION
FROM
CRE8TIVE DEVS

Cre8tive Devs. Software

Bitty Byte
Reactive & Event Driven
Cohort 1

CERTIFICATE OF COMPLETION