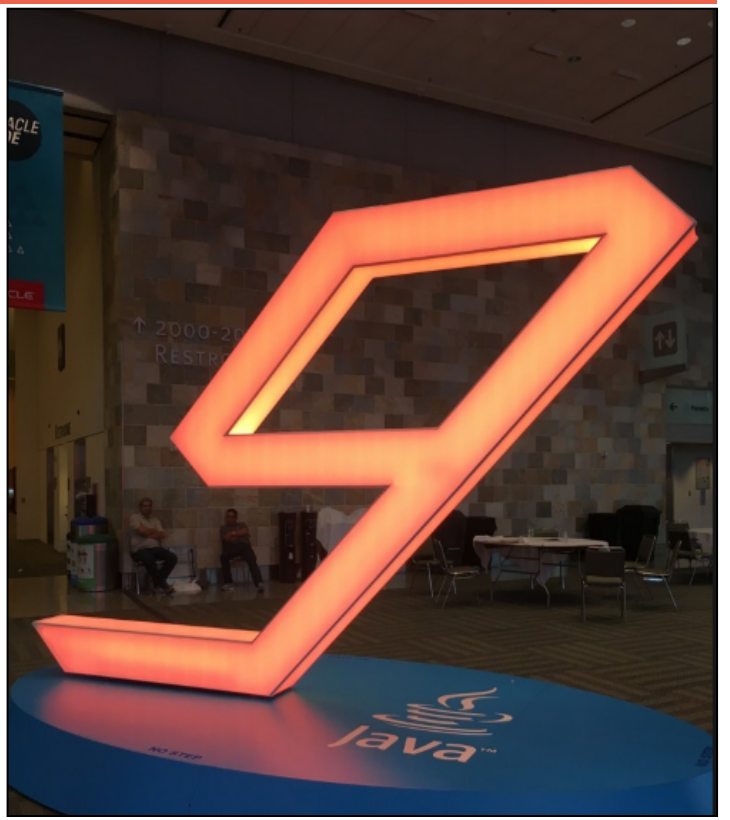
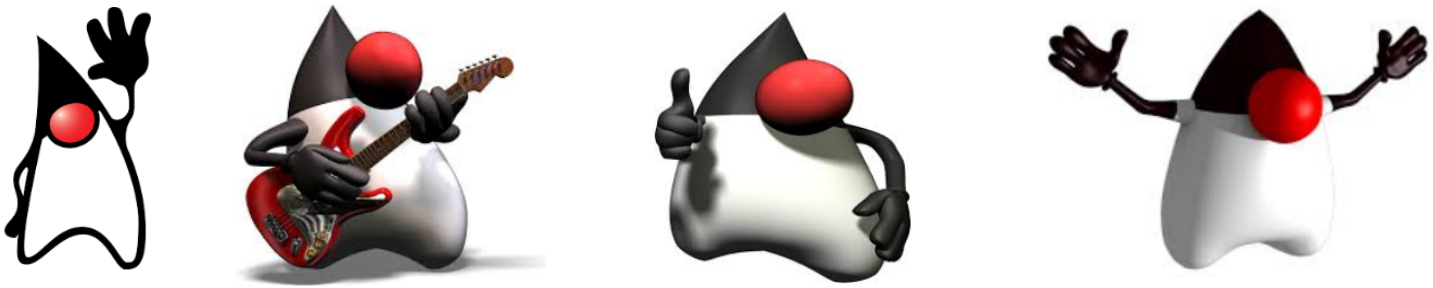
Java One – 2017 (Oct 1 - 5) – San Francisco ,CA





Subhash Koganti
skoganti@archmi.com

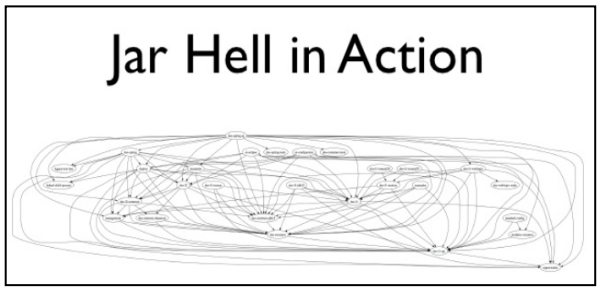
Buzzwords from Java One - OCT 2017

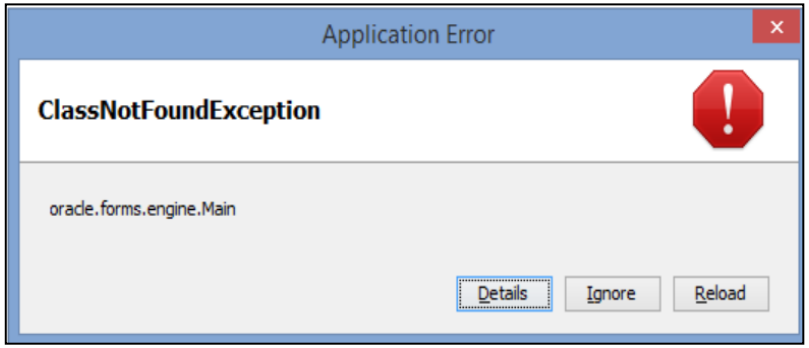
- Java 9 Modularity
- Java EE 8 / EE4J / Microprofile
- Server less Cloud Computing
- Micro Services/Cloud Native Applications/Containerization
- Reactive Programming
- Machine Learning
- Java For Mobile App Development

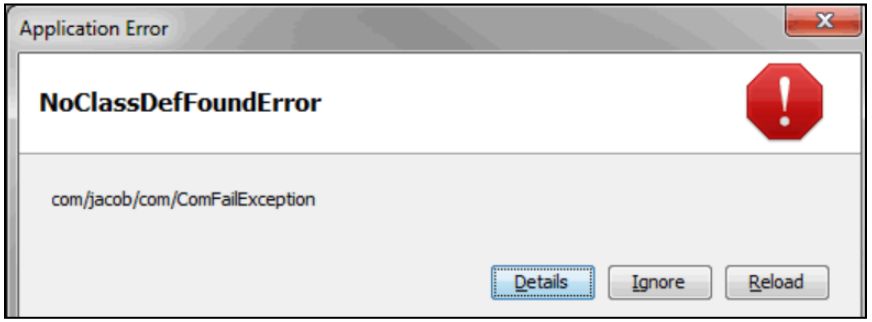
Java 9 Modularity – Problems with classpath

- Brittle and Hard to manage Runtime Class path in Java

- Monolithic Applications with heavy memory usage and vulnerable to fail easily







Java 9 Modularity – Monolithic & Brittle classpath

classpath:

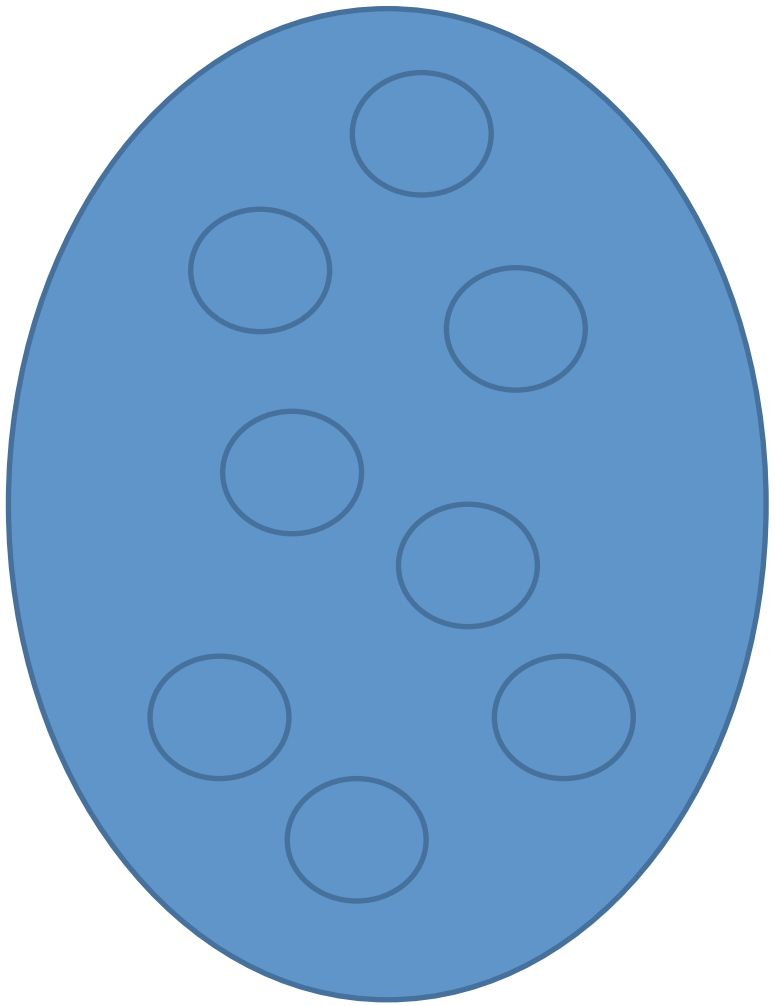
com.app.jar1,com.app.jar2,com.app.jar3,com.app.jar4,com.app.jar5,com.app.jar6,com.app.jar7,com.app.jar8,com.app.jar9,com.app.jar10,com.app.jar11,com.app.jar12,com.app.jar13,com.ap.jar14,com.app.jar15,com.app.jar16,com.app.jar17,com.app.jar18,com.app.jar19,com.thirdparty.jar1, com.thirdparty.jar2, com.thirdparty.jar3, com.thirdparty.jar3, com.thirdparty.jar4, com.thirdparty.jar4, com.thirdparty.jar4, com.thirdparty.jar6, com.thirdparty.jar7, com.thirdparty.jar8, com.thirdparty.jar9, com.thirdparty.jar10, com.thirdparty.jar11, com.thirdparty.jar12, com.thirdparty.jar13, com.thirdparty.jar14, com.thirdparty.jar15, com.thirdparty.jar16, com.thirdparty.jar17, com.thirdparty.jar18, com.thirdparty.jar19, com.thirdparty.jar20, com.thirdparty.jar21, com.thirdparty.dependency1.jar1, com.thirdparty.dependency1.jar2, com.thirdparty.dependency1.jar2, com.thirdparty.dependency1.jar3, com.thirdparty.dependency1.dependency1.jar1, com.thirdparty.dependency1.dependency2.jar1,com.thirdparty.dependency1.dependency2.jar1 com.thirdparty.dependency1.dependency2.jar2,com.thirdparty.dependency3.dependency2.jar4 com.thirdparty.dependency5.dependency2.jar6,com.thirdparty.dependency7.dependency2.jar8

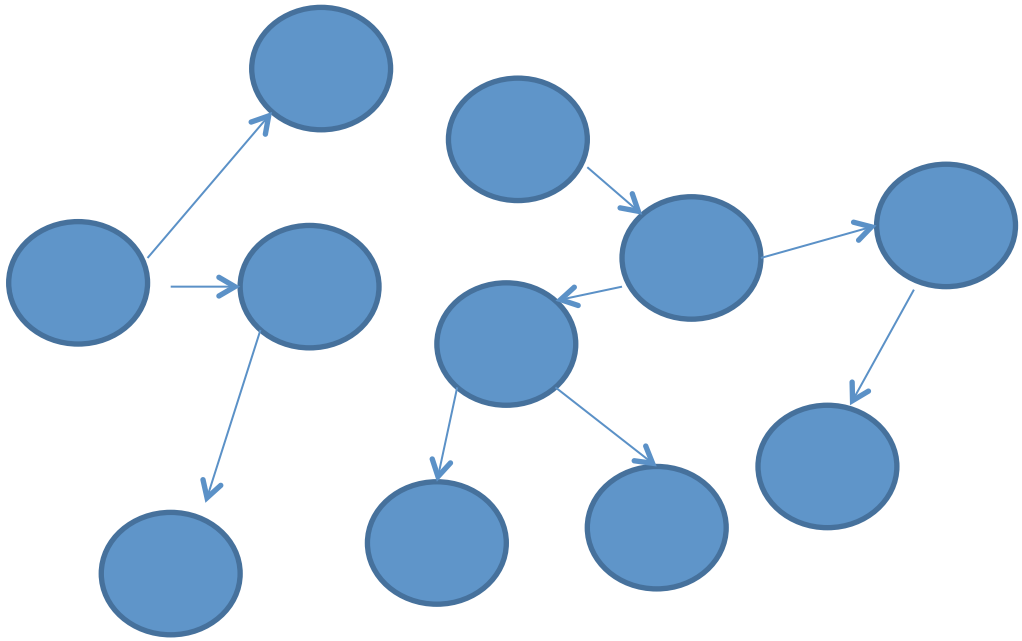
NoClassDefFoundError

Java 9 Modularity – How to solve the classpath problems ?

- We need Reliable Configuration
- We Need Strong Encapsulation and Abstraction
- We need ability to package what you only need and not other unnecessary stuff .
- We need to be able to conceal your international Implementations
- We need to be able to plug and play a feature / module
- We need to be able to subset a piece of the monolith and run it

Java 9 Modularity – Running the App in pieces instead of monolith

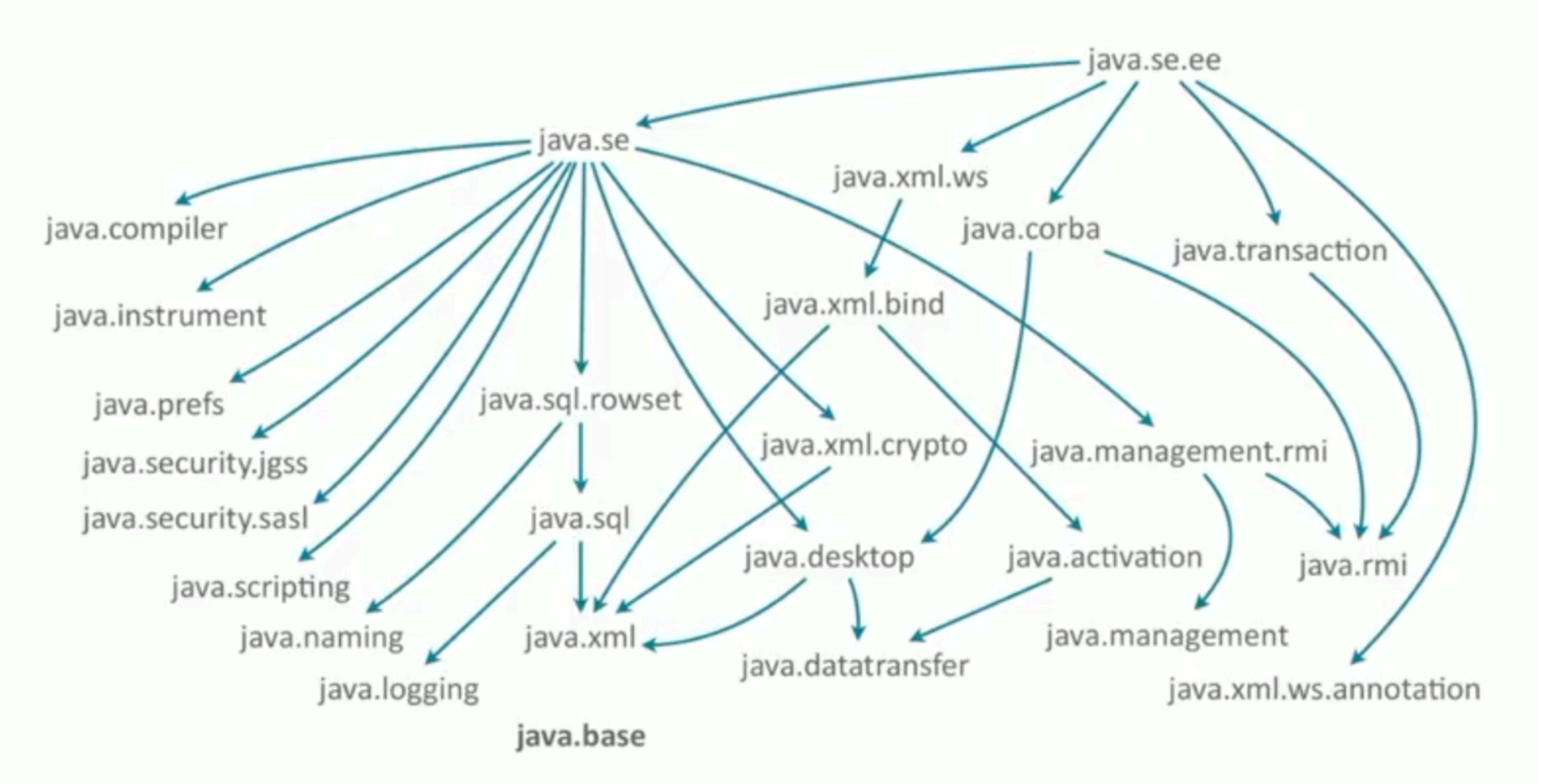




Java 9 Modularity - MODULES

- Our good old friend **rt.jar** is now removed from JDK.
- JDK Now has 94 modules out of which 26 are Java SE Modules .
- All of them fire up as modules in the **module path** in the JDK .
- Each Module has the following basic information defined :
  - What do I need to compile AND RUN
  - What do I expose to Other Modules so that they can Compile AND RUN

Java 9 Modularity – JDK Module Graph



Java 9 Modularity - MODULES

```
com.ratequote.provider/
└── src
    ├── com
    │   └── ratequote
    │       └── provider
    │           └── RateQuoteProvider.java
    └── module-info.java
```

module-info.java

```
module com.ratequote.provider {

    exports com.ratequote.provider;
    requires java.base;
}
```

```
com.ratequote.requester/
└── src
    ├── com
    │   └── ratequote
    │       └── requester
    │           └── RateQuoteRequester.java
    └── module-info.java
```

module-info.java

```
module com.ratequote.requester {

  requires  com.ratequote.provider;
  requires java.base;
}
```

Java 9 Modularity – MODULE Directives

The following are the module directives introduced in order to work with Modules and the module path :

- **requires** : specifies module A can read module B
- **requires transitive** – implied readability : specifies that module A can read module B and also any module that requires module A can also transitively read module B.
- **exports and exports … to** : specifies that one of the module's packages should be accessible to code in all other modules.
- **uses** : specifies a service used by this module – making the module service consumer.
- **provides with** : provides a service implementation- making the module service provider.
- **opens & opens .. to**: Allows runtime-only access to a package for example: to allow reflection at runtime like (Jackson/JAXB libraries)

Java 9 Modularity

- Basic Hello World Example
- Add exports and imports to a Module
- jlink
- jdeps
- jshell
- JDK Documentation and Module Graph

Java 9 Modularity – Other Significant Additions

- REPL (Read Evaluate Print Loop) is now available in java too through JSHELL

```
Subhash's MacBook Pro: ~/Subhash/JavaOnePresentation $ jshell
|  Welcome to JShell -- Version 9.0.1
|  For an introduction type: /help intro

jshell> String s = "Java One Demo"
s ==> "Java One Demo"

jshell> System.out.println(s)
Java One Demo

jshell> new java.lang.String().getClass().getModule()
$3 ==> module java.base

jshell>
```

Java 9 Modularity – Other Significant Additions

- Immutable Collections can now be created through Collection Factories

List.of()
Set.of()
Map.of()

Old way of Creating Collections :

```
List<RateQuote> qList = new ArrayList<>();
qList.add(new RateQuote("Q1234");
qList.add(new RateQuote("Q5678");
qList.add(new RateQuote("Q9012");
qList = Collections.unmodifiableList(myList);
```

Using Collection Factories :

```
List<RateQuote> qList =
            List.of(
            new RateQuote("Q1234"),
            new RateQuote("Q5678"),
            new RateQuote("Q9012"));
```

Java 9 Modularity – Other Significant Additions

- Enhancements to **Optional Class** introduced in JAVA 8:
  - **ifPresentOrElse**(Consumer action , Runnable emptyAction) :

    Similar to ifPresent , but if there is no Value, it executes emptyAction Runnable passed in .
  - **Stream() :**

    Returns a stream of 0 or one elements, depending on whether there is a value.

Java 9 Modularity – Other Significant Additions

- Examples of some enhancements to Stream API introduced in JAVA 8:
  - ofNullable

```
Stream.of("Q1234" , "Q5678" , "Q9012" )
    .flatMap(key -> {
          String quote = System.getProperty(key);
          return prop == null ? Stream.empty(): Stream.of(quote);
    })
    .findFirst()
    .orElseThrow(IllegalStageException::new);
```

Java 8

```
Stream.of("Q1234" , "Q5678" , "Q9012" )
          .flatMap( key ->
          Stream.ofNullable(System.getProperty(key)))
          .findFirst()
          .orElseThrow(IllegalStageException::new);
```

Java 9

  - takeWhile and dropWhile

```
List<RateQuote> quotes = quotesListInDescOrdrLnAmt
          .stream()
          .filter(quote -> q.getLoanAmount() >=450000)
          .collect(Collectors.toList());
```

Java 8

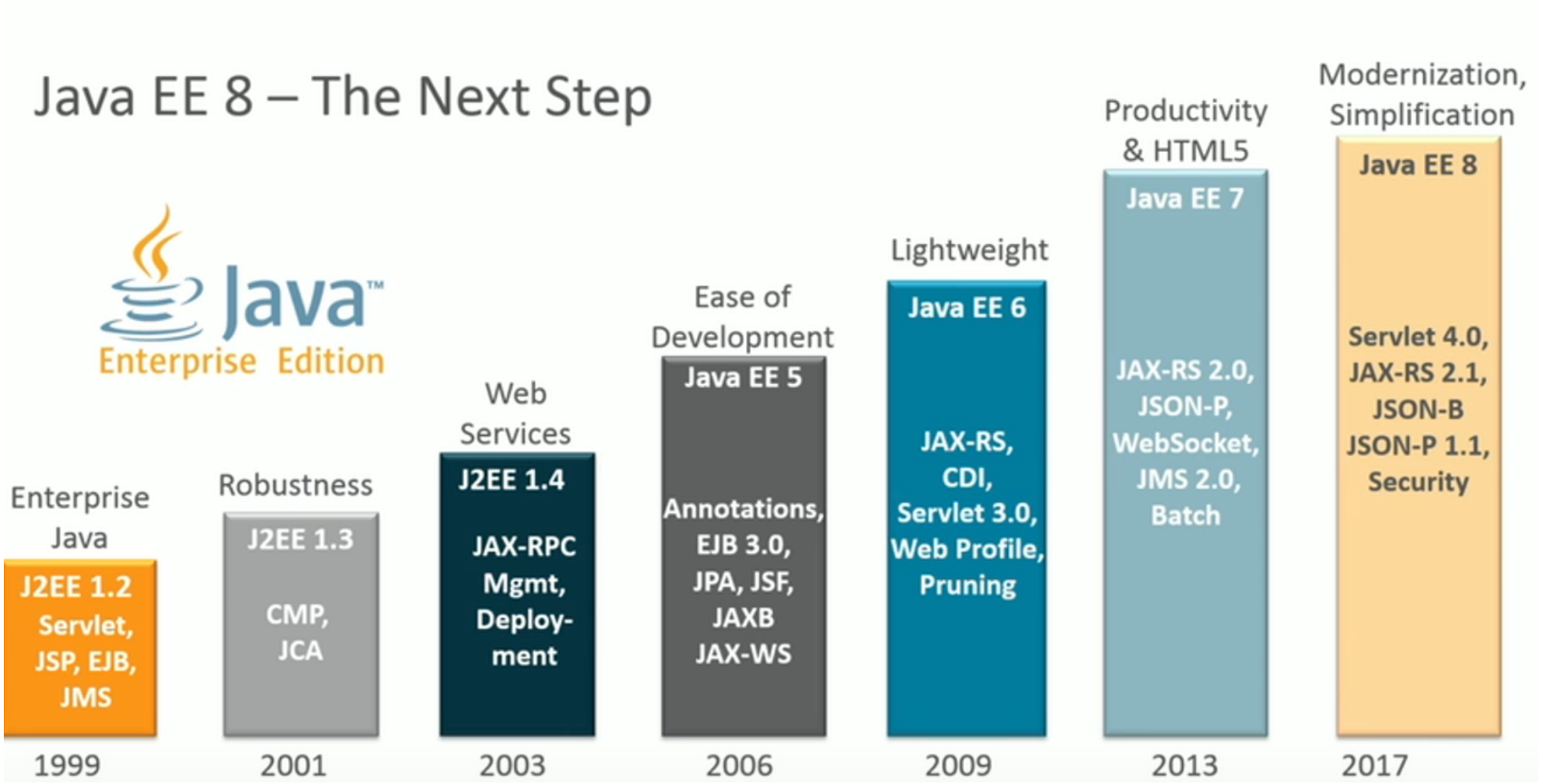
```
List<RateQuote> quotes = quotesListInDescOrdrLnAmt
    .stream()
    .takeWhile(quote -> q.getLoanAmount() >=450000)
    .collect(Collectors.toList());
```

Java 9

Java 9 Modularity – Adoption/Challenges/ Questions

- Is it right time to adopt java 9 ?
- How easy is it to understand and design an app in java 9 ?
- I have a lot of legacy code base.Is it easy to upgrade my legacy code base to Java 9 ?
- Is it fully backward compatible like the previous Java versions ?

Java EE 8 – Road to Java EE 8



Java EE 8 – Road to Java EE 8

Java EE 8 Community Survey 2016

- Surveyed community on technology importance
- Compared response to original Java EE 8 proposal
- Summary of findings
  - Focus on Servlet, REST, JSON
  - Accelerate delivery
  - Deprioritize new standards for Management, JMS, MVC

API from Original Java EE 8 Proposal	Survey Item	Importance	Revised Java EE 8 Proposal
JAX-RS 2.1	REST Services	High	No change, retain
Servlet 4.0	HTTP/2	High	No change, retain
JSON-B 1.0	JSON-B	High	No change, retain
JSON-P 1.1	JSON-P	Medium	No change, retain
CDI 2.0	Not surveyed	N/A	No change, retain
Bean Validation 2.0	Not surveyed	N/A	No change, retain
JSF 2.3	Not surveyed	N/A	No change, retain
Security 1.0	Not surveyed	N/A	No change, retain
Management 2.0	Management	Low	Withdraw Mgmt 2.0, Include Mgmt 1.0
JMS 2.1	JMS	Low	Withdraw JMS 2.1, Include JMS 2.0
MVC 1.0	MVC	Low	Withdraw

Java EE 8 – Road to Java EE 8

Java EE 8

JSRs – Original Plan 2014

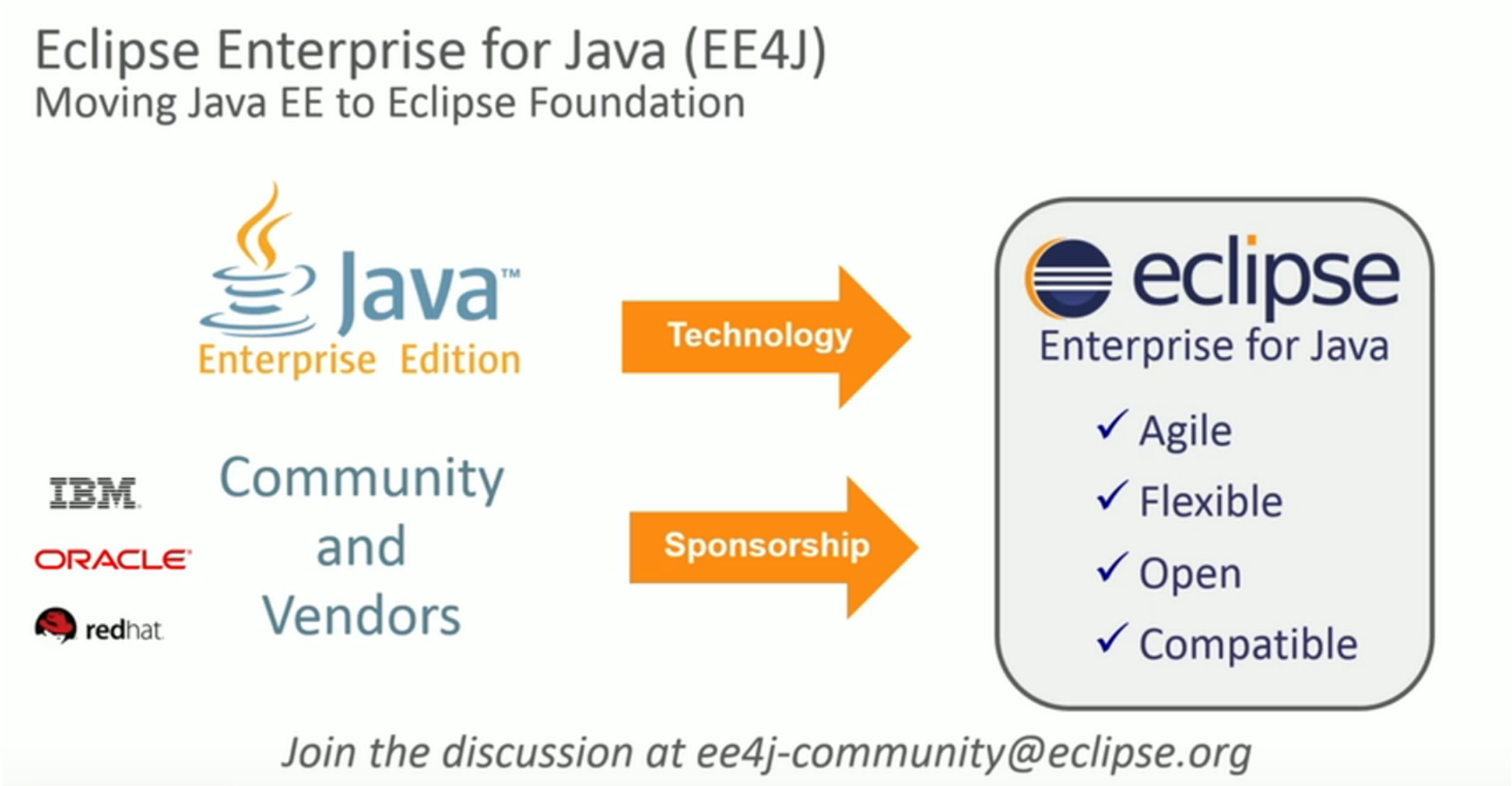
- Java EE 8 Platform and Web Profile
- Contexts and Dependency Injection 2.0 (CDI)
- Java API for JSON Binding 1.0 (JSON-B)
- Java Message Service 2.1 (JMS)
- Java Servlet 4.0
- Java API for RESTful Web Services 2.1 (JAX-RS)
- Model-View-Controller 1.0 (MVC)
- JavaServer Faces 2.3 (JSF)
- Java EE Management API 2.0
- Java API for JSON Processing 1.1 (JSON-P)
- Java EE Security API 1.0
- Bean Validation 2.0

Java EE 8

JSRs – Updated Plan 2016

- Java EE 8 Platform and Web Profile
- Contexts and Dependency Injection 2.0 (CDI)
- Java API for JSON Binding 1.0 (JSON-B)
- Java Message Service 2.1 (JMS)
- Java Servlet 4.0
- Java API for RESTful Web Services 2.1 (JAX-RS)
- Model-View-Controller 1.0 (MVC)
- JavaServer Faces 2.3 (JSF)
- Java EE Management API 2.0
- Java API for JSON Processing 1.1 (JSON-P)
- Java EE Security API 1.0
- Bean Validation 2.0

Java EE 8 – Road to Java EE 8



Java EE 8 – EE4J / Micro profile

- What is EE4J ?

Eclipse Enterprise for Java (EE4J) is an open source initiative to create standard APIs, implementations of those APIs, and technology compatibility kits for Java runtimes that enable development, deployment, and management of server-side and cloud-native apps.

https://projects.eclipse.org/projects/ee4j/charter

- What is Eclipse Microprofile ?

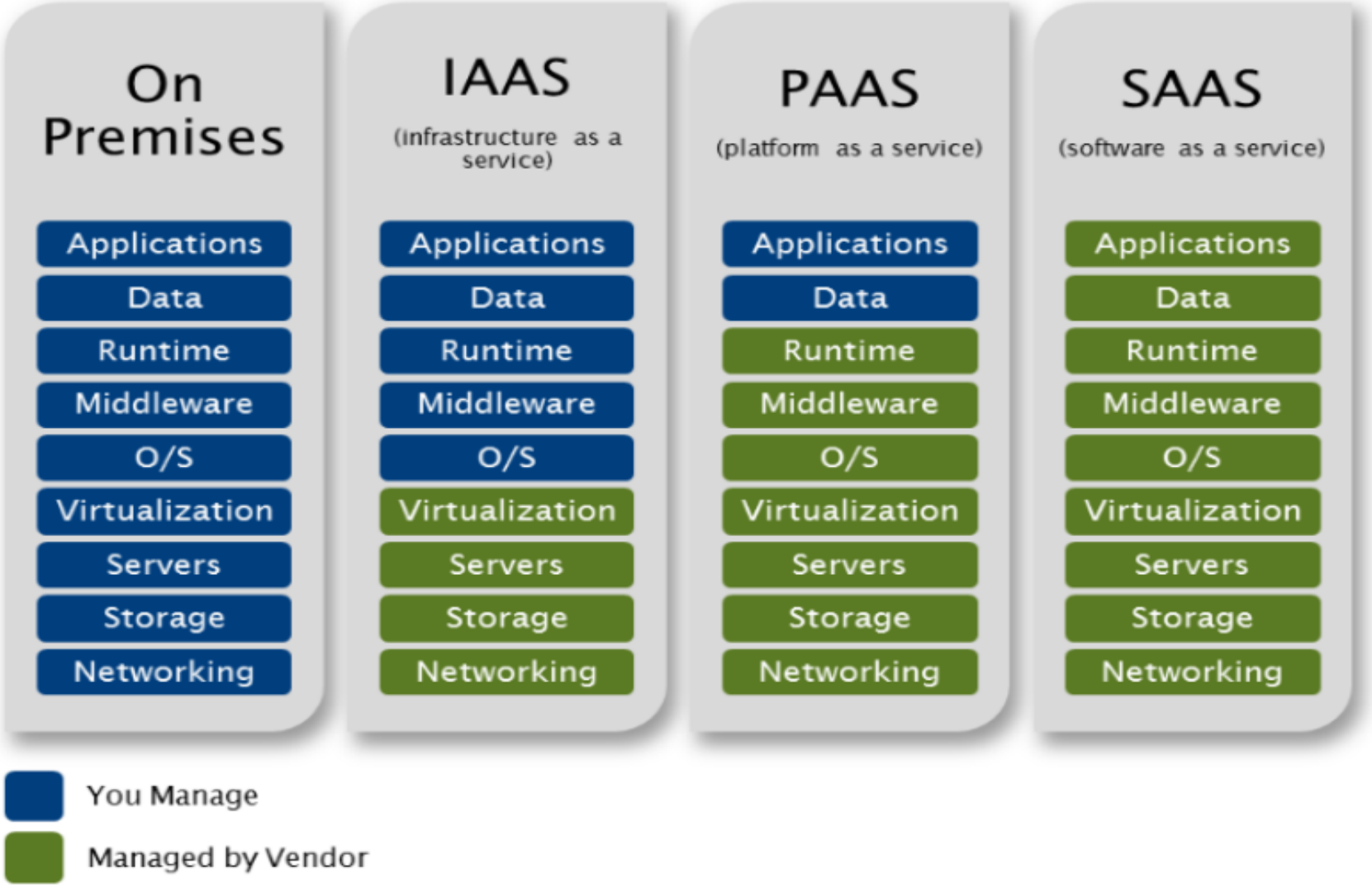
The microprofile.io community (http://microprofile.io/) is a semi-new community dedicated to optimizing the Enterprise Java mission for micro service based architectures.
The goal is to define a micro services app platform that is portable on multiple runtimes .

https://projects.eclipse.org/proposals/eclipse-microprofile

Java EE 8 – EE4J / Micro profile

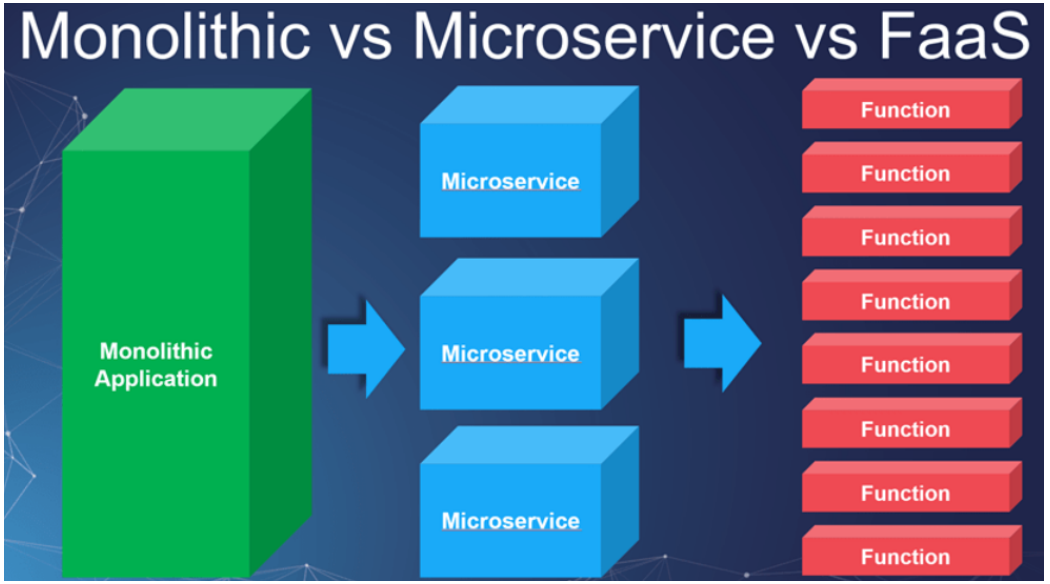
- Who are the Major Players in the eclipse micro profile ?

Serverless Cloud – Traditional Cloud stack of services



Serverless Cloud – FaaS (What is it ?)

- Does it not have any servers ?
- What is it and Who hosts it ?
- Who manages it ?
- Is there a subscription fee for it ?



AWS Lambda

Google Cloud Platform

CLOUD FUNCTIONS
Create small, single-purpose functions that respond to events in the cloud

Microsoft

Azure Functions

ORACLE

fn

Serverless Cloud – Oracle announces FN project

- Oracle joins the cloud wars for FaaS with FN project .

- FN is an event driven, open source , Functions as a service compute platform that
You can run anywhere . Its GIT repository is made public in Java One 2017 Key note session.

- The GitHub link for the open source project

https://github.com/fnproject/fn

- Written in Go

- Can be run in local with in minutes after reading the Github ReadMe doc for quick start.

Serverless Cloud – Helloworld on AWS Lambda

- Create a Simple Function in Java in Eclipse AWS tool kit
- Export it as a Jar file
- Upload the Jar on to AWS lamba using AWS management console
- Run a test and view the logs in AWS management console

Microservices – Wide spread adoption and maturing of microservices architectures .

The current Programming communities are widely adopting the micro services patterns in their application architectures.

Major patterns like Circuit Breakers , Service Discovery, Containerization are being adopted widely in the java community.

Examples of Frameworks that facilitate the use of micro services :

1) Netflix OSS (Open Source Software)

2) Spring Cloud with Netflix OSS

Explore Netflix Eureka and Netflix Hysterix dashboards to view their features.

https://github.com/Netflix/eureka
https://github.com/Netflix/Hystrix
https://github.com/spring-cloud-samples/eureka

Reactive Programming – Change in the programming paradigms and mindsets

- Traditional Imperative Style Programming :

  You tell the program WHAT to do and HOW to do .

- Functional Style Programming :

  You tell the program WHAT to do and the rest is taken care by a function/higher order function.

- Reactive Style Programming :

  It's a next step to the Functional Style programming . Where the programs react to the stimuli and execute the higher order function.

Reactive Programming –Example of Imperative Style

Find the Loan with highest RQI value in the year 2017 :

```
List<Loan> loans = createLoansList();
double highestRQI = 0.0 ;

for( Loan loan : loans) {

    if( loan.getLoanYear() == 2017 ) {

            if(loan.getRQI() > highestRQI ) {
                highestRQI = l.getRQI();
            }
     }
}
```

Reactive Programming –Example of Functional Style & Differences with imperative style

Find the Loan with highest RQI value in the year 2017 :

Imperative

```
List<Loan> loans = createLoansList();
double highestRQI = 0.0 ;

for( Loan loan : loans) {

   if( loan.getLoanYear() == 2017 ) {

            if(loan.getRQI() > highestRQI ) {
               highestRQI = l.getRQI();
            }
   }
}
```

- Every Value is associated with a variable that can change.
- Order of execution matters
- Repetition is controlled by programmer

Functional

```
List<List> loans = createLoansList();
double highestRQI = loans
            .filter( loan -> loan.getLoanYear() == 2017 )
            .map(loan -> loan.getRQI())
            .max();
```

- Every Value is associated only once and will never change
- Order of execution is not defined.
- Repetition is controlled by library

Reactive Programming –Lazy Evaluation in Functional Style Programming

```
public static void main(String[] args) {

        IntStream stream = IntStream.range(1,5) ;
        stream = stream.peek(i -> log("start-"+i)
                        .filter(i-> {
                                log("filtering-" +i);
                                return i%2 == 0 ;
                                })
                        .peek(i->log("after filter"+i);
        log("After the stream statement ") ;
        log(" The count in the stream is" + stream.count());

}
```

Output :

After the steam statement (Notice that This statement is printed first)
start -1
filtering-1
start -2
filtering-2
after filter-2
start -3
filtering-3
start -4
filtering-4
after filter-4
The count in the stream is 2

Reactive Programming –Next step to Functional style

The main pillars of the Reactive programming are :

1) They are elastic
2) They are Event driven / Asynchronous
3) They are responsive with lazy evaluation and respond only when its needed.
4) They are resilient.

Examples:

1) Excel sheet value calculations on update of a cell
2) Updating every mobile/webapp about the Stock price when it changes

Reactive Programming –
CompletableFuture (java 8) , RxJava –Reactive Extensions

```
public class HelloWorldCompletableFuture {

    public static void main(String[] args) throws InterruptedException, ExecutionException {


        CompletableFuture<String> bestTravelFuture
              = CompletableFuture.supplyAsync(HelloWorldCompletableFuture::findBestFlight)
                  .thenCombine(CompletableFuture.supplyAsync(HelloWorldCompletableFuture::findBestHotel), (s1,s2)-> s1+s2);

        System.out.println(“Executing Other OPERATIONs");
        System.out.println(bestTravelFuture.get()); // Blocking Call which waits for both threads to finish

    }
}
```

```
private static String findBestHotel() {

        System.out.println("Insider the findBestHotel");
        try {
            Thread.sleep(1000l);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("After calculating best Hotel ");
        return "Mariott";
    }
```

```
private static String findBestFlight() {

        System.out.println("Insider the findBestFlight ");
        try {
            Thread.sleep(2000l);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("After calculating best Flight");
        return "Delta - 507";
    }
```

Output:
Insider the findBestFlight
Insider the findBestHotel
Executing Other OPERATIONs
After calculating best Hotel
After calculating best Flight
Delta - 507Mariott

Mobile App Development using Java

Gluon :

A Mobile Application Development Platform to build Native Mobile apps using Java.

The Gluon product builds the native App based on the Device Type, by converting the java code in to the native app.

- Single App, Multiple Platforms (Based on Java – end to end)
- Drag & Drop, Rapid Application Development with Scene Builder for Java FX
- Hardware: Abstracted
- High performance
- Cloud Connected


Questions