

DevOps

Wednesday, August 23, 2023 10:23 AM

DEV+ OPS

TDD - Test Driven Development

1. Groovy - Java
2. Python

Operations -

- Linux - shell scripting
- Git + GitHub
- CI/CD - pipelines - Jenkins
- Docker
- Kubernetes
- IAC - Infrastructure as Code [Terraform + Ansible]

Monitoring Tools -

Splunk

Deployment -

AWS

Collaboration of Development Team and Operations Team

Groovy

Requirements -

JDK
Eclipse
JVM - interprets bytecode to native code executes the class file

Groovy is an object oriented language which is based on Java platform. Groovy 1.0 was released in January 2, 2007 with Groovy 2.4 as the current major release. Groovy is distributed via the Apache License v 2.0.

Object oriented language
Supports functional Programming
Supports Dynamic Typing - def age = 25

Waterfall Model

Requirement Analysis
System Design
Implementation
Integration and Testing
Deployment
Maintenance

JIRA - tool for planning

Demo seen

Agile

Scrum - holistic development approach
Follows a common for most of the projects. Incremental agile SD framework.

SCRUM Roles -

Product owner
Scrum Master
Scrum Team

Scrum Ceremonies

Sprint planning
Sprint review
Sprint retrospective
Backlog Refinement

Scrum artifacts

Product backlog
Sprint backlog
Burndown Chart

PRODUCT OWNER
SCRUM MASTER
SCRUM TEAM

Story points are added to mark a user story as important

Doable items at the sprint is called as sprint backlog

Picked from the product backlog.

Functional Programming -

Functional programming is a programming paradigm in which we try to bind everything in pure mathematical functions style. It is a declarative type of programming style. We can return a function, pass a function as a parameter etc.

GROOVY Day2

24-08-2023

Package - com.myapp.demo

Class file

First program -

```
package com.myapp.demo
```

```
println('My first Groovy Script')
```

Primitive Datatype - part of stack memory

User Defined Datatype or Reference Datatype - part of heap memory

Primitive Datatype

```
Int
Char
Byte
Short
Long
Float
Double
boolean
```

Reference Datatypes

```
Integer
Character
Float
Boolean
Long
```

Sort objects :

small case

```
package com.myapp.demo
```

```
def products = [
    newProduct(123, 'Iphone10', 434374),
    newProduct(456, 'Iphone11', 434355),
    newProduct(789, 'Iphone12', 434326),
]
```

```
def product = newProduct(123, 'Iphone10', 43434)
```

```
println products.sort{it.price}.productName
```

```
@Sortable(includes="price")
```

```
println products.sort()
```

```
@ToString(includes=["productName", "price"])
```

```
println products.sort({p1, p2 -> p1.price - p2.price})
```

Comparator - Compares the p1 and p2 and then return 1 if p1>p2 else return -1 and return 0 if all are same

MIN Max -

```
def products = [
    newProduct(123, 'Iphone10', 434374),
]
```

ArrayList -

```
def products = ['Iphone14', 'Oneplus11', 'SamsungFlip']
```

```
println products.class
```

Loop

```
for (product in products)
    println product
```

Closure:

```
{
    Parameter ->
}
```

Ex -

```
products.forEach{product->println product}
```

Or

```
products.each{println it}
```

We pass closure as a parameter and then call it using the **closure** keyword

```
def myFunc(val){
    val * val
}
```

```
println myFunc(myFunc(2))
```

```
def newFunc(val, closure){
    closure(val * val)
}
```

```
newFunc(100, {println it})
```

```
def cb = {println it}
```

```
def cb1 = {
    val -> println val
}
```

```
newFunc(100, cb1)
```

```
newProduct(456, 'Iphone11', 534355),
newProduct(789, 'Iphone12', 634326),
]
```

```
def res = products.findAll { product -> product.price >= 500000 }
println res.max()
println res.min()
println products.average {
    it.price
}
```

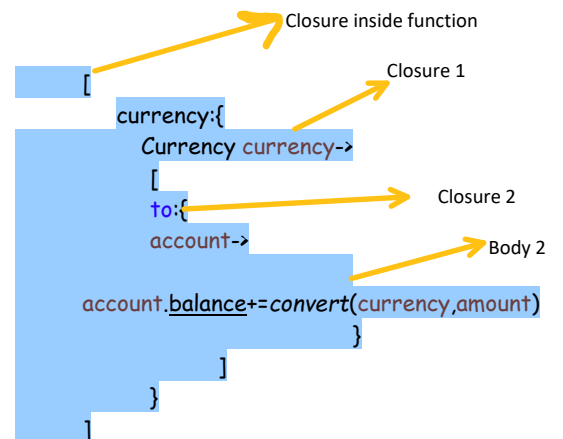
```
def stats = res.stream().mapToDouble { it.price }.summaryStatistics()
println stats.getMin()
println stats.getMax()
println stats.getAverage()
```

Reduction Step or reducing

DSL - Domain Specific Language

Ex -

```
LunchOrder {
    Soup: { "Sweet Corn" },
    Starter: { "Fried chicken" },
    MainCourse: { "Biryani" },
}
```



What is a closure?

Closures are anonymous in nature and can be assigned to a variable that can be executed later.
It is a code snippet.

```
package com.myapp.demo

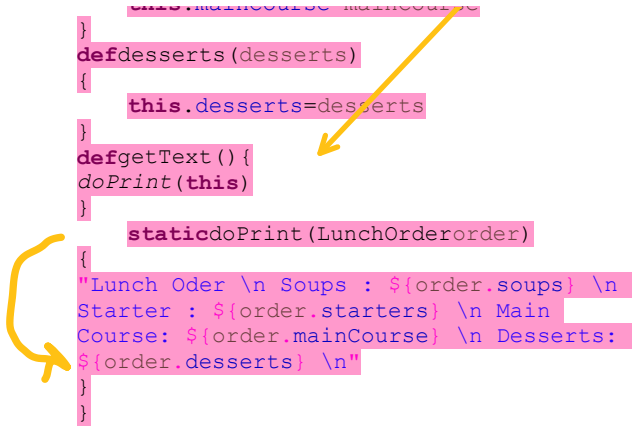
class LunchOrder {
    String soups
    String starters
    String mainCourse
    String desserts
    def meals = []
    static def create(Closure closure) {
        {
            LunchOrder lunchOrder = new LunchOrder()
            closure.delegate = lunchOrder
            closure()
        }
    }
    def soups(soups) {
        {
            this.soups = soups
        }
    }
    def starters(starters) {
        {
            this.starters = starters
        }
    }
    def mainCourse(mainCourse) {
        {
            this.mainCourse = mainCourse
        }
    }
    def desserts(desserts) {
        {
            this.desserts = desserts
        }
    }
}
```

```
println LunchOrder.create {
    soups "chicken soup"
    starters "baby corn"
    mainCourse "butter chicken"
    desserts "ice cream"
    text
}
```

```

    }
    def desserts(desserts)
    {
        this.desserts=desserts
    }
    def getText() {
        doPrint(this)
    }
    static doPrint(LunchOrder order)
    {
        "Lunch Oder \n Soups : ${order.soups} \n
        Starter : ${order.starters} \n Main
        Course: ${order.mainCourse} \n Desserts:
        ${order.desserts} \n"
    }
}

```



Python