

www.healthycodemagazine.com

Healthy Code

February 2015 ■ Vol I ■ Issue 11 ■ Rs.150



Interview with
Linda Rising
Agile Consultant

Microservices | Go Language | Spring Data Neo 4J



Follow us
on
Twitter

@healthycodemag

Healthy Code

February 2015

Contents

4

Microservices
*What are they and
why should you care?*

12

Code Jugalbandi

16

**Working with
Spring Data
Neo4J (SDN)**

22

**GrailsConf
2015**

26

**Interview with
Linda Rising**

34

**Effective Java with
Groovy**

38

Let's go with GO

Elancer Tech Group

Software Architecture

Through Examples, Case Studies and Exercises



GET PROMOTED - BECOME AN ARCHITECT!

Are you a developer or a designer aspiring to become an architect? Are you looking to get promoted as an architect or jump-start your career as a software architect? If you have answered "yes" to these questions, this training is certainly for you.

Contents overview:

- Design principles, patterns and architectural styles
- Creating an architecture (scenarios, NFRs, constraints, creation process, ...)
- Evolving an architecture (tools, techniques, refactoring, smells, ...)
- Living as an architect (skills required, technical leadership role, ...)

When: 28 Feb 2015 (Sat) - 10am - 6pm

Where: In a 4/5 star business hotel conference room in Outer Ring Road (near Marathalli) - will be announced to registered participants later

Investment:

- Early bird (Rs. 2,500),
 - Regular registration (Rs. 3,000),
 - Last minute (Rs. 3,500)
- [Cost includes handouts, buffet lunch & high tea]

Limited seats - so hurry up!

Trainer: Ganesh Samarthayam ([LinkedIn Profile](#))— well-known author and trainer

Details: <http://www.meraevents.com/event/software-architecture-ii> (register here)

For queries/clarifications: Hari Kiran (gharikir@gmail.com) / 96321 77909)

▶▶▶▶ From the Editor

Hello readers!!!

It's hard to believe, we are already into the second month of 2015.

We are back with another very interesting set of articles this month. The February issue is filled with loads of information about programming languages.

Rocky Jaiswal introduces us to the *Go* language throwing light on its capabilities and potential. We have our Naresha continuing to impress us about writing better Java code using Groovy. The Code Jugalbandi team is back with another melody and implementing it in a variety of languages.

Linda Rising a renowned speaker and author talks to us about the Agile Mindset. Linda, being such a fascinating person, the interview is sure to have an impact on you forever.

Rajesh Muppalla has written a fantastic article on one of the most happening architectures, the *Microservices Architecture*. This article talks about several issues with practical examples that will make you rethink the strategy of your projects.

Arun is back with Ravikant and his favorite Neo4j. They talk about using Neo4j with Spring framework.

And yes! Do not forget to try out the quiz on Java 8.

Enjoy reading and coding.



Prabhu





Microservices

What are they and why should you care?

Over the last year, Microservices has emerged as a “new” approach to building large scale distributed systems. This is an introductory article that provides background on Microservices, their evolution, the pros and cons and the state of the art, as it exists today.

You’ve recently joined a brand new client project as a lead developer/architect. You’re sitting in a meeting room and after having heard the client’s requirements, you are now contemplating the right architecture and technology stack for the project.

Your client wants you to build an e-commerce application for consumers. They want to support multiple clients including browsers on the desktop and mobile as well as native mobile applications. In those cases, your system receives an HTTP request in either *JSON* or *XML*, which executes business logic, accesses database and sends back response in *JSON* or *XML*. At the same time, the system should also expose an API to integrate with third party applications.

This sounds like a typical layered architecture, you say to yourself. You have built a couple of such systems in the past, so you decide to use the tried and tested architecture you are comfortable with. The different layers of your architecture are divided into Presentation, Business and Database components. For the technology stack, you have a choice between *Rails*, *Java*, *Scala*, *Nodejs*, *.NET* etc. It really does not matter what you use as this architecture pattern is language or platform agnostic. However, your client wants you to release an MVP (Minimum Viable Product) of the product in the next six months. In the end, you choose a technology stack that majority of your team of four developers are comfortable with.

Fast forward six months. You are at a team party to celebrate the success of the MVP which was delivered on time. The team celebrates the success but also reflects on what a small team of six (you added two developers mid-way during the MVP) have accomplished in a short time. The team is thankful to you for choosing the technology stack that they were comfortable with. The layered architecture also provided multiple benefits. It was very easy to make a change as all the code was in a single-source code

repository. For the same reason, it was easy for the two new developers to ramp-up and setup the entire application on their laptop, build it and run it locally within a couple of hours. The release was also painless given that there was a single deployment pipeline that would take the build from the CI (Continuous Integration) server and deploy it on a couple of load-balanced production boxes.

Fast forward two years later. After the success of the MVP and a few minor pivots later, the customer base of the product has grown. Your client also boosted their sales team in the last two years to aid the growth. And your team has grown to 20 developers. However, the product is not able to scale and evolve with the customer base. You realize that the application feels like a big ship that’s very hard to steer. The application has grown to thousands of lines of code. The ramp up of new developers is taking weeks instead of days. And you are finding it hard to look for developers who can work on your “legacy” codebase. As a result of all these factors, you are not able to deliver as fast as you did earlier to keep up with the business requirements.

Let’s try to understand what’s wrong with the application you built that’s challenging. First, give it a name. The term that is used for an application like the one above is: “**Monolith.**”

Below are some of the challenges associated with a large monolithic application.

- **Scaling issues** - On the one hand, a monolithic application can be scaled horizontally by running multiple copies behind a load balancer. But on the other hand, each component has its own scaling requirements as some components are heavy on I/O and some on memory and hence have different resourcing requirements. With a monolithic application, you have no choice but to provide the same (higher) resources to the entire application, potentially increasing infrastructure costs.

- **Technology changes and evolution** - Certain problems are best solved in certain technologies. The technology choices you made when you built the monolith may or may not be the perfect fit when your application evolves at a later stage. You might want to move to a different technology stack or replace individual pieces of the architecture. It could be for scaling needs or for better developer productivity. For eg: *JVM vs C++, SQL vs NoSQL*. With a monolithic application you are married to a technology stack for a long time.
- **Multiple teams** - Your teams are organized as UI, Services and Database layers. Doing any cross cutting features takes time and also needs approval for the right prioritization. Ideally, you want to organize teams around business capabilities to reduce the communication barriers so that an individual team can take things end-to-end without a lot of overhead. And if the teams are across locations or time zones it adds another level of complexity.
- **Continuous Delivery** - Continuous Delivery is about reducing the time it takes for a commitment made by a developer to hit production. It is hard to follow continuous delivery practices when you have a single large codebase. The cycle time will be larger because the entire application has to be built, tested and deployed for any change. In addition, each of the deployments is a high risk deployment.
- **Ramp-up of new members** - A large monolithic project is intimidating for new developers as it can take a long time to setup, build and run locally.

Is there a solution?

Yes. **Microservices**. A small autonomous set of services that work together, that can be scaled and released independently with different teams potentially using different languages across different locations.

So, how do we describe a Microservice?

Autonomous - Each service can be built using the appropriate tool for the job. Multiple teams of developers can independently deliver functionality in this model. Each microservice can have its own data storage using polyglot persistence.

Modeled around the business domain - Microservices should be vertically aligned to business capabilities to provide faster delivery of business goals and outcomes.

Small; does one thing and does it well - Where size is

a disadvantage for a monolith, it's an advantage for a Microservice. It should also follow the *Single Responsibility Principle* (SRP) and should strive towards loose coupling and high cohesion that are hallmarks of well-designed components.

Own build and deployment - A Microservice should be standalone and should be able to run independently from other systems and services.

Integrates via well-known interfaces - If the Microservices expose and talk using well-known interfaces using open protocols and standards, it provides technology heterogeneity, thereby allowing teams to use the technology stack that they are comfortable with or the best suited for the problem at hand.

What's with the name "Microservices"?

I heard the term "Microservices" the first time when Fred George spoke about them in his talk YOW 2012 - Brazil. Ronald Kuhn, the tech lead on Akka, thinks that the term Microservices is too ambiguous and likes to call them Uniservices instead, given on what they do.

Netflix used to call its implementation of Microservices as cloud native architecture and fine grained Service Oriented Architecture (SOA) but they have started to use the term Microservices since that has become widely accepted.

How is this different from Service Oriented Architecture (SOA)?

At the heart of it, both Microservices and SOA try to combat the challenges associated with large monolithic applications. SOA's approach uses protocols like *SOAP*, *RPC*, a complex middleware called *ESB* that meant different things to different people.

It had sound principles but there was a lack of consensus on how to do SOA. Moreover, the implementation itself was monolithic where dumb services were integrated with a smart ESB. Proponents of Microservices look at it as a specific approach to SOA in the same way that XP or Scrum are different approaches for Agile software development.

Why now?

There was always a desire to build systems this way (even in the SOA era) but the recent advances and maturity in the area of Cloud, Devops and Continuous Delivery have made it easier to build them right with Microservices being the building blocks.

Moving towards Microservices

There are several design aspects that one has to delve into and weigh the pros and cons of various approaches while moving towards Microservices.

Splitting the Monolith - The first consideration is around splitting the monolith. And the challenge here is to identify the appropriate service boundaries. In this context, Domain Driven Design (DDD) is a nice model to help break your system into Bounded Contexts that allow you to set logical and physical boundaries between your services. Another challenge will be around splitting the state. For most applications, that state will be the database. You should refactor your existing databases to de-normalize into one data source for table. You should also think about modeling foreign key relationships and transactions across two Microservices.

Inter-services Communication - You will have to decide the approaches for communication between Microservices. REST over HTTP is a sensible default choice for both synchronous and asynchronous communication. For the payloads, you have a choice of using JSON, XML or Binary Protocols. Prefer using JSON payloads for both external and internal service communication because of its readability

and integration across languages. Consider using Binary protocols like *Thrift*, *Avro* or *Protobuf* if performance is a consideration for your internal services.

Testing - Start with integration tests to make sure that implicit contracts between services are codified. However, these kinds of tests are hard to write and tend to be brittle when you have more than a handful of services. Use consumer driven contract tests in those cases to detect contract breakages. Testing should also be done in post-production environments by running fake requests and asserting on the responses. Consider using a tool like **Shadow** from **Twilio** while rolling out a newer version of a service to compare its responses with the existing service.

Deployment - Each of the services should be independently deployable for which you should prefer a single repository for each service. Follow Continuous Delivery principles ensuring that every change within the service is built and tested including integration tests as part of a CI pipeline. Upstream dependencies should trigger the contract tests pipeline. Once these go green, the changes should be



deployed in staging environments using a deploy pipeline. Use Continuous Delivery practices like Feature Flags and Canary Releasing to reduce the risk of new features and to roll back problematic code if required. Use Blue Green Deployments to ensure zero-downtime for your services. Investigate using container technologies like Docker with orchestration systems like Marathon to help manage complex deployment workflows and to auto scale based on load.

Failure Isolation - Failures are common when you have more than a handful of services. If Service A calls Service B which in turn calls Service C, a problem in Service C can cause service disruption to end users. Use Patterns like Timeouts, **Circuit Breaker**, **Bulk Heads** and Backpressure to isolate failures. In case of cascading failures, consider gracefully degrading the service rather than failing completely. The book **Release It** has a chapter on Stability Patterns which is a good introduction to this topic.

Service Discovery - This is the equivalent of the Service Registry problem in SOA. When you have multiple services, you need a way to discover other services that you are dependent on. DNS with a Load Balancer (LB) works for most cases, but is unable to scale in a dynamic environment where nodes are overloaded and are timing out and need to be removed from the LB or in cases where one needs to auto scale elastically according to load. **Zookeeper** has been a great choice in this area for some time as it provides the primitives to build a service discovery solution. **Consul** is emerging as an alternative that gives you all the things required for service discovery out of the box.

Monitoring - When you have multiple services talking to each other, you need tools that can aid in understanding the system behavior and diagnose performance issues. Aggregation becomes very important to provide a unified view of various systems and metrics. Consider using a single tool like **Logstash** or **Splunk** to query and aggregate logs from various Microservices. Distributed request tracing provides you with the visibility necessary from an end-to-end perspective when multiple services are involved. Use correlation IDs across various Microservices to track the flow of interaction between services and tie them back to the original request. Consider using **Zipkin** from Twitter, an open source solution inspired by **Dapper** from Google.

Documentation - If one does not know what your service does, it's likely that no one will use it. Use **Swagger** or **Postman Collections** to document your RESTful APIs.

API Gateway - If your clients need to orchestrate data from multiple Microservices, the API Gateway provides a way

to handle cross-cutting concerns like authentication, quota management, rate limiting, basic analytics etc. You can also provide a different granularity for end users by composing one or two Microservices using the API Gateway. **Zuul** from Netflix is an open source alternative and **3-scale**, **Apigee** and **Mashery** are commercial solutions.

Microservices OSS Stack

Let's talk about some OSS that's available to help to migrate towards Microservices.

- **Netflix** - Netflix has a fully functional Microservices stack that they have been using internally for a few years. They have open sourced several key pieces of their stack in recent times. Some notable libraries include:

- Hystrix - Fault Tolerant library
- Eureka - Service Registration and Discovery

If you are using Spring, the Spring Cloud Netflix project integrates these libraries to the Spring ecosystem.

- **Twitter** - All of Twitter's traffic till end of 2012 was served by a single monolithic rails application, internally called "**MonoRail**." Since then they have moved to using Twitter Server for building their Microservices, which is built on top of **Finagle**. **Finatra** and **Finch** are noteworthy frameworks from Twitter. **SoundCloud** and **Tapas** are examples of companies that are successfully using Twitter Microservices stack in production.
- **Typesafe** - Typesafe's reactive platform provides the building blocks to build a Microservices platform. You can use Spray (to be super-ceded by Akka-HTTP), Akka-Camel, Akka-Streams to build your own Microservices stack on top of the Typesafe platform. Bench.co is a company that's using the Typesafe stack to build their Microservices platform.
- **Dropwizard** - This is a JVM based microcontainer that pulls together battle-tested libraries for configuration, metrics, logging to create RESTful Microservices.
- **Roll Your Own** - Of course, you could also roll your own stack by using various pieces from existing stacks and building on top of them.

The above list is not comprehensive and is focused on JVM platform as majority of the Microservices at my company, Indix, run on the JVM. We are currently using Dropwizard and the Typesafe stack but are also actively evaluating the Netflix and Twitter stacks. We also have non-JVM

Microservices where we are using Bottle and Flask for Python, Express for Node.JS and Sinatra for Ruby.

What are the disadvantages?

The rosy picture I painted after the initial gloom might make you believe that Microservices would solve all your architectural problems. Sorry to break your heart, but like everything else in software, Microservices are not a silver bullet. It's an architecture style that is still maturing and has its own share of challenges.

- When you are running multiple Microservices, it becomes extremely important to manage the operational complexity associated with it. You need to make sure that you have the right tools in place to visualize and orchestrate your deployment and also ensure that you are able to monitor the complex interactions between these systems. You need to build the right set of tools as well as the skill set within your teams to handle this complexity.
- If your team is using multiple languages and technology stacks to build their services, a lack of standardization might slow you down especially when debugging failures.
- As you work with implicit interfaces across Microservices, you need to worry about backward compatibility and versioning. It needs co-ordination across teams and the solution to this may or may not be technical.

Should I move to a Microservice architecture?

This is one of those questions that has a YMMV (Your Mileage May Vary) answer.

At Indix, we started with a monolith (but modular) application, which was built over the course of the last three years. As I write this article, we are migrating to Microservices across all our teams. The reason for us to move to Microservices is captured in the section describing the problems with the monolith. In hindsight, starting with a monolith and breaking it down into Microservices with the team growing and the system getting complex seems like the right approach.

To give a different perspective, I should refer to Go-CD – an open source build and release management tool. I worked on the team about three years back. Right now, it's more than a six-year-old project and the code-base is still a monolith. I think the reason why it has stayed so is because there were not more than 10 developers at any point in time. The

team is co-located and would ramp up only two or three new developers every year. The team uses techniques like Branch by Abstraction and Strangler Approach to evolve with changing technological trends. It will be interesting to see if the community decides to split Go-CD into multiple Microservices at some point.

If you want to move towards a Microservices architecture, you should read Martin Fowler's article on Pre-requisites for Microservices.

Summary

Microservices is an important tool in the arsenal of an evolutionary architect. Microservices themselves have evolved out of the hard-earned lessons of engineers building large-scale systems. When systems and teams scale, breaking them into Microservices decouples your systems and gives more options and choice to evolve them independently. Like any other tool, it is important to know when and how to use it. Hopefully, this article has provided you with enough context to decide on both.

Further Reading

If you are really interested to dive deeper into building Microservices, I suggest ordering a copy of the excellent book by Sam Newman on **Building Microservices** (as of January 2015, it's still in Early Access).



Rajesh Mupalla is the co-founder and Director of Engineering at Indix. He leads the data platform team that is responsible for collecting, organizing and structuring all the product-related data collected from the web. Prior to Indix, he was a technical lead on Go-CD, an agile and release management product at ThoughtWorks. In his earlier stint at Thoughtworks, Rajesh served as a technical lead on multiple customer projects in the domains of e-commerce and social media. He is passionate about big data, large-scale distributed systems, continuous delivery and algorithms. While excelling at these areas, Rajesh also gives back by mentoring and coaching developers in pursuit of building better software.

Healthy Code SUBSCRIPTION

Read the best authors!

Write the best code!

Create the best products!

Share the joy of programming!

Subscribe for Healthy Code

Subscription Period	Amount
1 month	Rs. 150 /-
3 months	Rs. 400 /-
6 months	Rs. 775 /-
12 months	Rs. 1500 /-

Visit our website for subscription:
<http://www.healthycodemagazine.com>

You can send us a cheque in the name of Healthy Code.

Address: 11/6, First Floor, Fourth Street,
Padmanabha Nagar, Adyar, Chennai 600020
Ph: 044-42337379, Cell: 91-98842-94494

Happy Reading!

Scala and Java 8

Workshop and Corporate Training

Also, the latest trainings on Mobile and Javascript Toolkits

Scan to visit us
on the web



durasoftindia.com

info@durasoftindia.com or 91-98842-94494



Code Jugalbandi

In 2013, Dhaval Dalal was inspired by Jugalbandi, an Indian classical music form in which musicians have an improvised musical conversation, each using a different instrument, while exploring a shared musical theme together akin to Jazz in some ways. Dhaval thought, "What if we could have a conversation about some programming concepts in this way? Instead of musical instruments, what if we use different programming languages?"

This led to the first Code Jugalbandi between us. Code Jugalbandi inspired and fuelled our learning by focussing on dialogue and exploration. More languages in the room meant more perspectives too. Exploring a paradigm through many languages gave us a better map of the territory than any single language could. Above all, Code Jugalbandi was a playful way to learn together!

Creator and Listener

We met regularly with over a period of several months, exploring some key issues related to Functional Programming (FP). This culminated in a live Code Jugalbandi at the Functional Conference in Bengaluru, 2014. During the FP Code Jugalbandi, we explored nine **themes**, which became **melodies**. Similar to the musical Jugalbandi, there are two roles: **creator** and **listener**. We called the creator of the melody **Bramha** and the one who listens and responds with a different instrument, **Krishna**. + In the last Code Jugalbandi Article, we looked at the melody, the expression problem. This time we will look at **Sequencing**.

The Sequencing Problem

Lets say we have a sentence and we want to capitalise words that have size less than or equal to 3.

Sequencing - Ruby

Bramha: In Ruby, in an imperative style, with mutation of variables, it would look like this

```
words = "all mimsy were the borogoves"
split_words = words.split("\s")
caps_words = []
split_words.each do |w|
  ;; enumeration and filtering
  cap-words.push(w.capitalize) if (w.size < 3)
end
words_s = caps_words.join("\n")
```

Bramha: In the above code, enumeration and filtering are interleaved. Let's try and separate enumeration and filtering.

```
words = "all mimsy were the borogoves"
words.split("\s").map do |w|
  w.capitalize
end.filter do |w|
  (w.size < 3)
end.join ("\n")
```

Bramha: This feels a bit more structured, we're chaining operations, one after the other. This style is called sequencing,

where we're moving data down processing pipelines.

Sequencing - Clojure

Bramha: In Clojure, we could chain functions together through function nesting.

```
(def words "all mimsy were the borogoves")
(def using-nesting
  (join "\n"
    (filter (fn [s] (< 3 (count s)))
      (map capitalize
        (split words #"\"s")))))
(println using-nesting)
```

Bramha: This is hard to understand. Deep nesting leads to many parentheses in Clojure that is considered a code smell in Clojure. Let's try and lose a few parenthesis through function composition.

```
(def using-composition
  (let [f (comp
    (partial join "\n")
    (partial filter (fn [s] (< 3 (count s))))
    (partial map capitalize)
    (fn [s] (split s #"\"s")))]
    (f words)))
(println using-composition)
```

Bramha: The awkward thing about these styles of coding is that the order of functions seem back to front. The arrow syntax sidesteps composition in lieu of a *threading-macro*.

```
(def using-arrow
  (->> (split words #"\"s")
    (map capitalize)
    (filter (fn [s] (< 3 (count s))))
    (join "\n")))
(println using-arrow)
```

Krishna: BTW, why is it called a *threading-macro* in Clojure? The threading word in there makes me think of process threads.

Brahma: The word *threading* is used in the sense of threading a needle through cloth, not to be confused with process threads! Since it is implemented as a macro, is is called a *threading-macro*.

Sequencing - Scala

Krishna: Lets see that how can we achieve the same in Scala.

I'll first show using chaining.

```
val words = "all mimsy were the borogoves"

println(words
  .split(" ")
  .map(w => w.toUpperCase)
  .filter(w => w.length <= 3)
  .mkString("\n"))
```



Ryan is a software developer, coach and advisor, based in Cape Town. He has been working with code for more than 15 years. Ryan assists individuals and teams to manage the evolution and complexity of their software systems. Ryan is a passionate learner and enjoys facilitating learning in others.



Dhaval a hands-on developer and mentor, believes that software development is first an art and then a craft. For him writing software brings his creativity to the fore. His interests are in architecting applications ground up, establishing environments, transitioning and orienting teams to Agile way of working by embedding himself within teams.

You can follow them on Twitter @CodeJugalbandi

Krishna: Now, I'll define a few function types.

```
val split = (s: String) => s.split(" ")
val capitalize = (ws: Array[String]) => ws.map(_
  .toUpperCase)
val filter = (ws: Array[String]) => ws.filter(_
  .size <= 3)
val join = (ws: Array[String]) => ws.mkString("\n")
val seqCapitalized = join compose filter compose
  capitalize compose split
seqCapitalized (words)
```

Krishna: Just like in Clojure nesting and composition examples, this is flowing against the grain of thought. Let me have the cake and eat it too. I can make use of Scala's **andThen**.

```
val seqAndThened = split andThen capitalize andThen
  filter
  andThen join
seqAndThened (words)
```

Sequencing - Groovy

Krishna: In Groovy too, one can quickly define closures like this.

```
def split = { s -> s.split(' ') as List }
def capitalize = { ws -> ws.collect
  { it.toUpperCase() } }
def filter = { ws -> ws.findAll
  { it.length() <= 3 } }
def join = { ws -> ws.join('\n') }
```

```
def seqComposed = join << filter << capitalize << split
println (seqComposed(sentence))
```

```
def seqAndThened = split >> capitalize >> filter >> join
println (seqAndThened(sentence))
```

Sequencing - Racket

Brahma: In Racket, one can achieve this using Rackjure (a Clojure inspired ideas in Racket by Greg Hendershott, <http://www.greghendershott.com/rackjure>).

```
(require racket/string)
(define s "All mimsy were the borogoves")

(define (capitalize s) (map (λ (w) (string-upcase
  w)) s))
(define (words<=3 s) (filter (λ (w) (<= (string-length
  w) 3)) s)) (define (sentence->words s) (string-split
  s))
(define (join-words ws) (string-join ws))
```



```
(define sentence-with-3-words
  (compose join-words capitalize words<=3 sentence-
>words)) (sentence-with-3-words s)
```

```
; using threading from Rackjure
(require rackjure)
(require rackjure/threading)
(~> s sentence->words words<=3 capitalize join-
words)
```

Reflections

Bramha: In whatever functional programming language you're using, if you find yourself deeply nesting functions, you can be sure there is a better way of "threading" functions together. Moreover, if you are going against the "grain of thought", look for a way in the language to go with the grain!

Krishna: This style is also known as **concatenative** programming where function composition is the default way to build subroutines. Functions written in concatenative style neither represent argument types nor the names or identifiers they work with; instead they are just function names laid out as a pipeline, in such a way that the output type of one function aligns with the input type of the next function in the sequence. In this way the order of function application gets transformed into order of function composition.

Bramha: One of the immediate benefits of this style is that it makes the code more succinct and readable. We can express some domain processes very clearly in this way, and we can easily extract sub-sequences into new functions to further clarify our statements about the domain.

Krishna: This style is useful whenever we are processing data through a sequence of transformations, and it is surprising how much of our domain code is trying to do just this!

References

Learn more from <http://codejugalbandi.org>

Healthy Code

February 2015

Vol 1 | Issue 11

Publisher:

T. Sivasubramanian

Editor:

S. Prabhu

Production:

New Horizon Media Private Limited

Proof Reader:

Shweta Gandhi

Publication Address:

11/6, First Floor,
Fourth Street,
Padmanabha Nagar,
Adyar, Chennai - 600020

Printer: Sakthi Scanners (P) Ltd

7, Dams Road, Chindadripet,
Chennai - 600002.

Contact Phone Number: 044-42337379

Email:

siva@healthycodemagazine.com

Web Site:

<http://healthycodemagazine.com>

Price: Rs. 150 /-

Disclaimer:

The articles and contents of the Healthy Code magazine are sourced directly from the authors with their permission. Healthy Code doesn't assume responsibility for violation of any copyright issues by an author in their respective article. We have made sure to the best of our knowledge that the articles and code presented here are without any errors. However, we can't be held liable for any damage or loss that may arise from using these.

- Publisher

Working with Spring Data Neo4J (SDN)



Spring framework provides various modules to build robust enterprise applications through the basic implementation of Object Oriented design principles and follows a layered approach of component oriented architecture. Spring-dao module simplifies RDBMS operations with a powerful abstraction over JDBC, JPA and ORM frameworks like Hibernate. On the other hand, while NoSQL databases gained significance and relevance, a dedicated umbrella of projects under the name **Spring-data** were formed. Spring-data is a uniform, standard approach to work with different types of data stores. Spring Data JPA, Spring Data MongoDB, Spring Data Solr, Spring for Hadoop and Spring Data Neo4j (SDN) are few of the projects under the Spring Data banner. In this article, we'll discuss the SDN module.

SDN bootstrap

SDN project provides Object-graph support and repositories for Neo4j. You may know that in ORM relational entities (tables and relationships) are mapped to objects. Similarly in SDN, graph entities (nodes and edges) are mapped to objects. SDN provides APIs to perform various graph operations to manage and manipulate the graph stored in Neo4j repository.

You can setup SDN easily using **maven**. Build tools like maven simplifies our tasks to hunt for the specific versions of required libraries and helps us in automating the complete build life cycle of a project. Please refer to the SDN site for more build-tool specific configurations. Let's add SDN to an existing Spring-based web application. Add Spring data Neo4j dependency to maven's pom.xml file. This will download the immediate dependency on SDN and all its dependent libraries transitively.

```
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-neo4j</artifactId>
  <version>3.1.0.RELEASE</version>
</dependency>
<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
  <version>1.0.0.GA</version>
</dependency>
```

In Spring's application context XML file, we have to add the following SDN namespace to be able to use the SDN tags as part of them. We will also configure the data store using `storeDirectory` attribute.

```
<beans xmlns="http://www.springframework.org/
schema/beans" ... xmlns:neo4j="http://
www.springframework.org/schema/data/
```

```
neo4j" xsi:schemaLocation="... http://www.
springframework.org/schema/data/neo4j http://
www.springframework.org/schema/data/neo4j/spring-
neo4j.xsd">
...
<!-- Activate Spring Data Neo4j repository
support -->
  <neo4j:repositories
    base-package="com.healthycode.repository" />
  <neo4j:config base-package="com.healthycode"
    storeDirectory="target/
healthycode.db" />
  ...
</beans>
```

Neo4j locks the given store directory when it is running. You cannot access it simultaneously through the standalone Neo4j server and the SDN API in embedded mode.

Problem Statement

Let us build a small application to maintain entities in the Healthy Code journal. The graph shown in Figure 1.0, illustrates the nodes and edges in the journal graph.

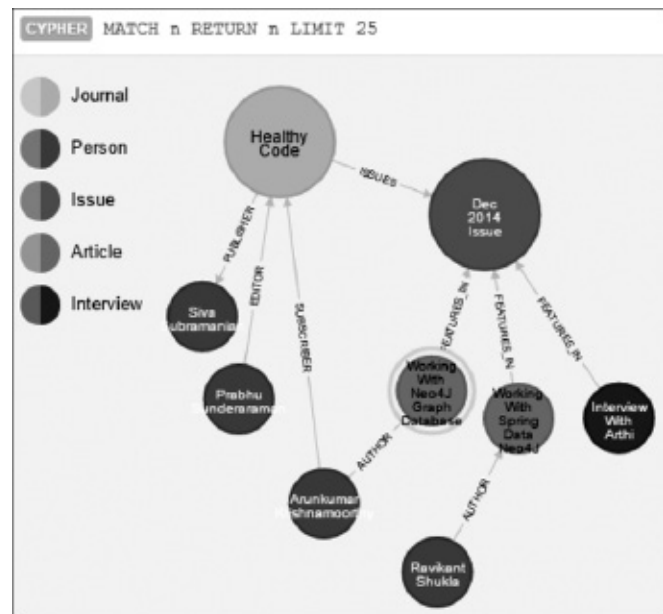


Figure 1.0. HealthyCode graph

The Healthy Code journal is issued every month. Various persons play different roles in Healthy Code. While it has one *PUBLISHER* and one *EDITOR* role, it has many *SUBSCRIBERS* and monthly *ISSUES*. Each monthly issue features several articles and interviews. Each article has one or more authors. Let's use the SDN API to perform CRUD operations and execute different queries on the graph.

Object Graph Mapping - Modelling Nodes and Relations

SDN API provides various annotations to map a Java object with nodes and relations in the graph. All POJO classes that map to a node in the graph should have `@NodeEntity` annotation. Here is the *Person* node modelled as a POJO along with SDN mapping.

```
@NodeEntity
public class Person {
    @GraphId
    private Long id;
    private String name;
    private String email;
    private String address;
    private String mobile;
    private String profileSummary;

    //getters and setters go here
    ...
}
```

Here is the *Article* node modelled as a POJO along with SDN mapping. As an article can have one or more authors, we have associated *Article* with a set of *Person* nodes with the relationship type as *AUTHOR*. Here, *AUTHOR* is a simple relationship wherein it does not have any properties.

```
@NodeEntity
public class Article {
    @GraphId
    private Long id;
    private String title;
    private String summary;
    private String tags;
    @RelatedTo(type = "AUTHOR",
        direction = Direction.OUTGOING,
        elementClass = Person.class)
    private Set<Person> authors = new HashSet<Person>();
    //getters and setters go here
}
```

Here is the monthly *Issue* node that gets published. One or more articles and one or more interviews can feature in the issue. As two different types of nodes are sharing the same relation with the *Issue* node, the attribute *enforceTargetType = true* ensures that both the relationships are cascaded at the time of persisting an *Issue* node.

```
@NodeEntity
public class Issue {
```

```
    @GraphId
    private Long id;
    private String name;
    private String coverPage;
    private Integer volumeNo;
    private Integer issueNo;
    private Date issueDate;
    @RelatedTo(type = "FEATURES_IN",
        direction = Direction.OUTGOING,
        elementClass = Article.class,
        enforceTargetType = true)
    private Set<Article> articles = new HashSet<Article>();
    @RelatedTo(type = "FEATURES_IN",
        direction = Direction.OUTGOING, elementClass =
        Interview.class, enforceTargetType = true)
    private Set<Interview> interviews = new HashSet<Interview>();
    //getters and setters go here
}
```

Let's create the root *Journal* node with all the relationships. You may have to pay specific attention to the annotation `@RelatedToVia` that relates a *Person* with a *Journal* through a relationship entity *Subscriber*. We will discuss about it in the next section. Just remember that a *Node* can directly have the related nodes or it can have the relationship itself.

```
@NodeEntity
public class Journal {
    @GraphId
    private Long id;
    private String title;
    @RelatedTo(type="PUBLISHER",direction=Direction.
    OUTGOING)
    private Person publisher;
    @RelatedTo(type="EDITOR",direction=Direction.
    OUTGOING)
    private Person editor;
    @RelatedToVia(direction=Direction.OUTGOING)
    private Set<Subscriber> subscribers = new
    HashSet<Subscriber>();
    @RelatedToVia(direction=Direction.OUTGOING)
    private Set<Issues> issues = new
    HashSet<Issues>();
    //getters and setters go here
}
```

Let's talk more about the annotations that we have used so far and also some more in SDN API.

- **@NodeEntity:** Maps a Java class to a node in the graph. All fields in the Java class (primitive or convertible to *String* using the built-in conversion service) are by

default mapped to properties in node.

- a. In case you want only some of the fields to be mapped to node properties, use attribute *partial=true* along with the annotation. This will map only those fields with *@GraphProperty* annotation to the node properties.
- b. SpringDataNeo4j includes a custom conversion factory that comes with converters for *Enums* and *Dates*. Transient fields are not persisted.
- **@GraphId**: Maps a *long* field to the id of the node or relation.
- **@GraphProperty**: Explicitly maps a field in Java class to a property in node or relationship. This is relevant when the entity annotation has the partial flag set to true. Default values are specified as String representations and will be converted to the correct target type using the existing conversion facilities.
- **@Fetch**: This annotation indicates SDN to eagerly load an associated node entity or relationship entity.
- **@Indexed**: Annotated field or type will be indexed. Index can be used at the time of retrieval. This annotation has additional attributes to specify the type and level of index. Often an index is used to establish the start node for a traversal.
- **@Labels**: Neo4j uses labels to identify the type of the node. While using SDN, it automatically creates a label

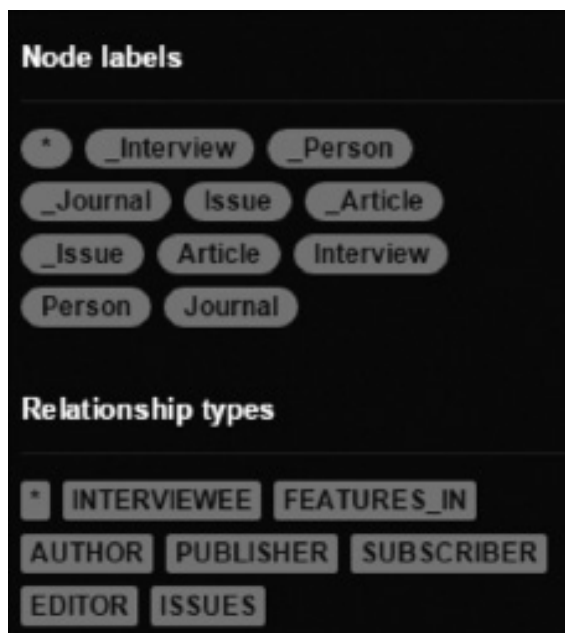


Figure 2.0. Node labels along with SDN generated labels with underscore prefix

with an underscore(_) prefix to the class name as shown in Figure 2.0. SDN uses this to map a node retrieved from the graph to appropriate type mapped in Java.

- **@Query**: In case we need to associate a node, say X, with an iterable list of nodes directly or indirectly related to this node X, in the graph dynamically, we pass in a cypher query to this annotation along with required parameters. SDN will do the rest by executing the query and exposing the obtained collection of nodes through the annotated field.
 - a. The provided query must contain a placeholder named {self} for the current entity. Additional parameters are taken from the parameter's attribute of the *@Query* annotation.

```
@Query(value = "start n=node({self}) match (n)-[r]->(article) where r.type = {relType} return article",params = {"relType", "AUTHOR"})
```

```
private Iterable<Article> myArticles;
```

- **@RelatedTo**: Associates one node entity with another node entity along with the direction and type of relationship.
 - a. For 1:1 relationship, this annotation is optional. When a value is set in the annotated field, relationship is created. When the value is null, the relationship is removed. An example is *An editor in the Journal node*.
 - b. Collection based one-to-many relationships return managed collections that reflect addition and removal of the underlying relationships. When the collection has to be modified, field should be of type *Set*. For read-only collection field should be of type *Iterable*. An example is *Set of articles in an issue*.
- **@RelatedToVia**: Associates one node entity with another node entity via a *Relationship* entity. The annotated field represents the *RelationshipEntity* discussed in the next section. Example is a *Journal* having a set of subscribers related via *SUBSCRIBER* relationship containing details of subscription.
- **@GraphTraversal**: If you feel the above mappings are insufficient, you can implement your own graph traversal logic and link that class as an attribute of this annotation.

Relationship Mapping

A relationship with zero or more properties is a powerful

feature in Neo4j. It comes handy when we write queries to filter nodes or traverse from one to another along desired relationships. A relationship is categorized by a *type*, a *start* node and an *end* node. Start node and end node imply the direction of the relationship. Relationships can have an arbitrary number of properties. Declaring a separate POJO class to model a relationship is optional. It is more often required only when using properties as part of relationships. In our example, a *Journal* node is related to an *Issue* node via *ISSUES* relationship. *ISSUES* relationship type is mapped to *Issues* relationship entity like this.

```
@RelationshipEntity(type = "ISSUES")
public class Issues {
    @GraphId      private Long id;
    @StartNode    private Journal journal;
    @EndNode      private Issue issue;
    private String issueDate;
    // getters and setters go here
}
```

The annotations, *@Fetch*, *@Indexed*, *@Labels*, *@Query*, that we discussed in the context of a node entity are still applicable within a relationship entity for the same purposes.

CRUD with GraphRepository API

SDN comes with various typed repository implementations that provide instant access to almost all common graph operations at no extra code. You get features of all the repositories by extending *GraphRepository<T>*. You may have noticed the Neo4j configuration in Spring's application context file where the support for Neo4j repository was added along with the base package.

```
<neo4j:repositories      base-package="com.healthy
code.repository" />
```

Following code snippet illustrates *JournalRepository* in our solution.

```
public interface JournalRepository extends
GraphRepository<Journal>{ }
```

We don't have to provide any implementation of repository. We just have to wire it with any Spring class with *@Transactional* support as follows.

```
@Service
@Transactional
public class JournalService {
    @Autowired
    private JournalRepository
        journalRepository;
```

```
    public boolean saveJournal(Journal
journal) {
journalRepository.save(journal);
return true;
}
}
```

The concepts like dynamic proxies, CGLIB, code instrumentation are put into picture to create classes implementing the repository interface at runtime and inject an instance of it like other normal beans. *GraphRepository* has all the required methods to save, delete and find graph entities. In addition, it has a method to pass a cypher query along with parameters to query the graph. For instance, in the following code snippet we are trying to fetch an *Issue* for the given date.

```
public Issue    getIssueForDate(Date    issueDate)
{
    Map<String, Object> params = new HashMap<String,
Object>();

    params.put("issueDate", issueDate);

    Result<Issue> issueForDate = issueRepository.
query( "MATCH (i:Issue) where i.issueDate =
{issueDate} RETURN i",params);    return
issueForDate.singleOrNull();
}
```

In case you have a complicated query result that contains more than one type of result columns, then consider using the following *@Query* annotation through a method in the repository interface. The following example fetches article and the issue it has featured, for the given author name.

```
public interface PersonRepository
    extends GraphRepository<Person> {
    @Query("MATCH (p:Person)-[:AUTHOR]-
(article:Article)-[:FEATURES_IN]-(issue:Issue)
WHERE p.name={0} RETURN article, issue")
    List<ArticleIssue> getArticlesAndIssuesForAu
thor(String authorName);
}
```

At run time, the place holder *{0}* will be substituted by the first parameter passed during the method invocation. The class *ArticleIssue* is a user defined value object that maps the query result to a POJO. Here is the definition of the value object with the *@QueryResult* annotation that maps the columns in the query result to fields in the POJO.

```
/**
```



```

* VO representing a graph query result that
fetches a pair of article and the
* issue in which the article has featured
*
* @author arun
*
*/
@QueryResult public class ArticleIssue {    @
ResultColumn("article")    private    Article
article;    @ResultColumn("issue")
private Issue issue;    //getters and setters
go here
}

```

Neo4j Template

If you have worked with Spring-Jdbc or Spring-Hibernate modules, you would have come across *JdbcTemplate* or *HibernateTemplate* respectively. These template classes are one-stop solution for almost all operations with the data store. On similar lines, SDN provides *Neo4jTemplate* as a convenient API to perform any graph operation across the whole graph. Unlike *GraphRepository* API where you need to create a separate repository interface for each entity, *Neo4jTemplate* allows you to directly invoke the operations. *Neo4jTemplate* instance is automatically injected in any of the Spring managed beans. SDN uses the *base-package* attribute in *neo4j:config* to scan and inject these standard dependencies.

```
@Autowired private Neo4jTemplate neo4jTemplate;
```

Once you have *Neo4jTemplate*, you can access all their straightforward APIs to perform various graph operations. Below code snippet shows accessing one of the methods exposed by *Neo4jTemplate* and how to retrieve entities returned by the query using *Result* the class.

```

public List<Issue> getIssuesWithTemplate() {
    Result<Issue> result = neo4jTemplate.findAll(Issue.
class);
        List<Issue>    issues    =    new
ArrayList<Issue>();
        for (Issue i : result) {
            issues.add(i);
        }
        return issues;
}

```

Neo4jTemplate exposes a suite of APIs to completely maintain and manage the Neo4j graph right from creating Node with attributes to creation of Relationships, Indexes, etc. We will expect you to go through the API of *Neo4jTemplate* and try

out individual operations.

Conclusion

In this article we discussed how to map a POJO with entities in the Neo4j graph viz. *NodeEntity* and *RelationshipEntity*. SDN API manages most of their functionalities via annotations. We also used both *Repository* and *Neo4jTemplate* to perform basic CRUD operations to execution of query and processing the result. We hope that this article would have got you started with Spring Data for Neo4j API.

References

- *Good Relationships – The Spring Data Neo4j Guide Book*, Michael Hunger
- <http://spring.io>
- <https://github.com/ramselabs/Healthycode-Spring-Data-Neo4J.git>



Arun is a hands-on Solution Architect for the complete stack in Enterprise applications, application integration, and android application development. After a decade of industrial experience in various Software development in leadership role, Arun founded Ram Software Engineering Labs in 2010 offering consulting on specific phases of software development, corporate training on diverse technologies and software engineering. Arun and his team at Ram Labs have also built a collaboration platform for educational institutions where they have heavily used Neo4j.



Ravikant Shukla is a Software Engineer at Ram Software Engineering Labs. He spends most of the time building web applications on Java Enterprise stack. As part of one of the product development he integrated underlying Neo4j data store with his web based application.

GrailsConf 2015

Grails Conference in India



The New Year started with a bang for the Grails developers when **GrailsConf 2015** took place at The India Habitat Centre, New Delhi on Saturday, 10th January. The one-day conference showcased a lineup of eminent speakers with over 160 participants attending the event. The gathering was mostly Java and Groovy developers, though I noticed many Java developers who were amazed to see the power and brevity of the Groovy language.

Here's a very brief rundown on all the sessions and speakers from the GrailsConf 2015.

Highlights:

The day started with a keynote by **Dr. Venkat Subramaniam**, an award-winning author, founder of Agile Developer, Inc. Titled **It Could Be Heaven Or It Could Be Hell**, Venkat motivated developers towards productive and sustainable development practices. Few key points that he mentioned which I really liked are:

- **Languages mould our thoughts** – The way we design the software is greatly influenced by the languages we know. So learn more languages. Don't stick to one or two. When we know multiple languages, it would change the way we code in our primary language. He also quoted *Be a Polygot Programmer* - By Neal Ford.
- How fast we can learn depends on how frequently we are exercising our mental muscles, i.e. investing on learning new things on a regular basis.
- Unit testing is to software like exercise is to human body. So write more and more unit test cases preferably following TDD.

One could clearly sense the excitement in audience after his keynote and I was one of the excited ones.



The next session was **Grails Goodness – The Search Is Over** by **Roni C Thomas and Manoj Mohan** from *IntelliGrape*. They demonstrated how easy and fast it was to develop with Grails. Their talk focused on the audience who are fairly new to Groovy ecosystem. Topics that were briefly discussed included:

- **GVM**: Groovy environment manager tool to move to different versions of Grails, with a simple *gvm* command.
- **Multiple datasources**: Just configure multiple database settings in configuration file, and use that bean anywhere to talk to another database.
- **Scaffolding**: Generate CRUD dynamically with just a single statement, which gets regenerated automatically when the domain changes.
- **REST API**: Creating controllers to render XML/JSON response with least effort using scaffolded code.
- **Plugins**: Use Spring Security plugin for securing the application, which is a ready-to-use feature.



Kunal Dabir, Senior Consultant with ThoughtWorks, spoke about **Demystifying Gradle Build Scripts**. His talk focused on:

- Defining custom tasks, i.e. the extra behavior.
- Domain Specific Languages (DSL)
- Showing how scripts work internally and covered operations like *apply* and *task*.
- Hidden handlers that can be used to find methods that are available.
- Task API and its internals.



Manoj Thakur, Director Engineering – Flipkart gave a presentation on **ElasticSearch**. In general, he outlined usages and key concepts allowing the participants to get meaningful analytics from the machine generated data, which grows rapidly and in real time. He shared a few configurations that worked really well for them like the unicast discovery mode. Then many people from audience shared their experience of working with ElasticSearch, making it an interactive talk.

Naresha K, a technologist with Channel Bridge Software Labs, spoke on **Effective Java with Groovy** where he recommended better practices. He mentioned that if once these are followed then it would make life much simpler and they included using more annotations, better handling of null objects, closures and delegate concepts.

We moved onto the next session by **Shrikant Vashistha**, Director Engineering – GlobalLogic, who spoke on **Specification By Example In Grails With Geb And Spock** and discussed how a specification should be written. He mentioned that the aim of writing spec is not only testing but also shared understanding and documentation. Shrikant introduced the concept of *Definition of Ready*, which is when the story should be given to the developer. It should at least be 80% ready. He also discussed the benefits of GEB and took us through a few examples. Page object concept has definitely made it simple to write and reuse the code.

Finally, the conference had a panel discussion specifically on upcoming trends in the JVM Landscape where many interesting questions were discussed by the panelists, which gave us a lot of clarity on topics like Java 8, Lambda Expressions. The panel talked about how to decide which tool/technology is right and some good practices.

After an hour of networking lunch, Dr Venkat Subramaniam was back with a session on **Applying Groovy Closures For Fun And Productivity**. His talk revolved around the concept of *Closures*, *Curry*, *Trampoline* and *Memoize*. I could have never understood those concepts better without his excellent examples and the use cases he shared. It was actually real fun learning them.

My huge thanks to all the fantastic speakers for taking out time to share their knowledge and experience with all of us. Also, many thanks to all participants whose company I really enjoyed. I feel really excited and I am ready for the GrailsConf 2016.



Amit Jain is a tech lead with more than 7 years of experience in Java and Grails development. He is an Agile enthusiast, a certified Scrum Master and an author of Remote-Pagination Grails plugin. These days he is enjoying learning Scala.



Promote your Technology event with Healthy Code Magazine

- User Group Events and Meet-ups
- Hackathons and Code Retreats
- Programming Bootcamps and Workshops
- Product and Technology Demonstrations
- Technology Conferences

Reach out to the Programming Community!

Just send us an email at siva@healthycodemagazine.com and we will give the promotion you need!



Interview with Linda Rising

It's great talking to you Linda. Let's start with a politically incorrect question. How old are you?

Hahaha. I am 72. In a lot of talks, I tell the audience how old I am. I tell them that I am not an exceptional person. I was never a major genius. I never made significant contributions to Computer Science. I only went to ordinary state schools and not MIT or Stanford. Whatever I do can be done by anybody. I encourage people especially when you have a dream. They lose the dream, because they think are old or may be they are not smart enough. And in agile mindset talks, I emphasize that there are two ways of looking at the world. The agile way is to believe that you can do better tomorrow, if you are determined. If you have a dream and you are looking at doing it, you should do it. It doesn't matter how old you are.

I don't care how old you are and don't mind telling people how old I am.

Could you please formally introduce yourself to the Indian audience?

Right now I live in Nashville, Tennessee, a little state in the US close to the East Coast. I have been living there for the last three and half years since my husband retired. Prior to that, I lived in Phoenix, Arizona for 22 years. I have been an independent consultant for the last fifteen years. I started doing that because I was laid off in the company I was working. After I got laid-off I

moved to Denmark and started working in a Danish company. Over there, I was surprised to find out that people in Europe were inviting me. I was invited to go to UK, Germany, etc., to give talks, help them with retrospectives, do a little bit training about patterns and I thought this works out very well. I thought when I had to go back to US, I would need to get a proper job. But what was surprising was that I could continue doing what I did in Europe in US as well and that was in year 2000. So, I would like to call myself an independent consultant. I don't know if I would be doing this next year or not. But I am doing it right now. I won't know if people would call me for training or invite me for talks a year from now. So, I take one year at a time.

I talk about and provide consulting on *Patterns*, *Retrospectives*, *Fearless Change* - the book I have written, introducing new ideas in organizations. My primary interest is in how the human brain works. I am primarily concerned about software development and making it better. I am definitely interested in Agility. But I generally like to talk about anything that can make anybody lead a better life.

Fantastic. Just curious! How many times have you talked about yourself so far?

Haha, I don't know. I have lost count. But the way I say it, may be make you feel like you are telling it for the first time. Oh, I would like to say things differently every time so that it sounds different.





You have been a professor, speaker and author. Have you been a programmer?

Oh. Yes. I have worked on defense projects. Worked on 777 airplane. I worked on telecommunications. I have worked for small, medium and large sized companies. I have written lot of software. On the airplane project, I was **Ada** Guru. So do you know about the Ada programming language? It is named after a woman. She was the first woman programmer. It was designed for safety critical domains like avionics. So it was used in systems concerned about saving lives, worried about explosions in defense. The whole 777 airplane was written using Ada. It was a new programming language in the early 80s. People were resistant to using it. People were scared to learn it because it was complicated and my job was to help people learn Ada. Ada is not an object-oriented language. It's object based and it has something like a package that was new for people used to **Fortran**, **C** or **Pascal**. It is an intermediate language on the way to object-oriented languages. It was big paradigm shift for people at that time. So, I have done lot of programming.

You have a violent past, don't you?

Oh. Yes. I have worked on defense projects. Worked with weapons. Written code about finding targets.

You have played several roles. Which role do you enjoy the most -- programmer or speaker or author?

I have enjoyed all of them. I have learnt from every one of these. Everything I have done is used in what I am doing now. I often talk to people who say that they are wasting their time either on the job that they don't like or working in something that no one will ever use. They think they are wasting their time when they should be raising their children and spending time with the family and they say that part of their life is worthless. My message is that everything you do will be used in everything you do later. Nothing goes waste.

The reason is I started my life as a chemist. I worked in a research lab and not in computer science or software engineering because when I graduated in 1963 those jobs were not available. I thought I wanted to be a research chemist. And, unfortunately, I was wrong. I was interested in Biochemistry and I learnt to my dismay that what Biochemists do is kill animals. That's what I did for six months. I know many ways of



killing rats. More ways than you can possibly imagine. Finally, one morning I ran to the fourth floor where you pick up animals and I just couldn't do it. I couldn't put my hand into the cage and see that animal coming towards me without knowing that my job was to kill it. So I quit my job. But, I think what ever I learnt about killing animals as a Biochemist is helping me now. I learnt a lot from that experience. You should never believe that this part of my life is worthless. You will always use something from that part. You will always be better in what ever you decide to do later on. It all hangs together. *John Muir*, founder of sierra club in US, said, '*If you pick up anything in the Universe you find it hooked up to everything else*'. So we can never isolate an experience and say, ah, that's worthless and we don't need to pay any attention to that. All the things are connected together. Don't you think that's true? I think that's very true. So everything you do is precious. And every experience is precious.

Who introduced you to Agile?

In the early 90s, I was working in a telecom company. Its main business was working on a switch - Big switches that are used for long distance phone calls. So those switches had 8 to 9 million lines of code. Everybody in the company knew that it wasn't sustainable and the business was shrinking. People were moving to newer technologies and they knew that they would make long distance phone calls using landlines and the switch business was dying.

So, they started lot of small experimental projects with the idea that one of them would take over the business. The switch is a big, heavy-duty process and requires CMM Level 4. People started on the smaller projects and they tried to use the same heavy-duty process and it didn't work. *Many of the smaller projects had no requirements*. We didn't know how to begin. We were trying to create a requirements document with no real requirements. I was doing the research on Patterns and teaching how to use them (like the Gang of Four Patterns) and my boss came to me and said, ' Linda, may be there are some patterns for doing those small projects, with a small team and no requirements.' He asked me to find a way.

I came across *Kent Schwabber* and there were people writing about his approach to projects. At that time, there was no book for that and only a website was available. It was called <http://controlchaos.com> and it is still there. There was a big picture on that website

with Rugby players that meant nothing to me. I started reading, sent him some emails, and I said, 'Can you please tell me about this?'. I asked, 'Can we do this? What is Scrum? I am trying to read your website, but I have lot of questions'. He was very helpful and got me started. He sent me information and answered my questions.

I put together a little presentation on what is scrum. I had the picture of the Rugby players huddling. I started with my little teams. I told them I don't know if this will work. But, we could always try it. We can see if it can help us in some way and we will just learn from that experience. The teams were very desperate. And they started doing experiments on Scrum. They were used to really heavy- duty work and suddenly it was all about short iterations. It was all about sitting down, talking to customers, which they had never done before.

The customer came and said, we don't know what we want, but we want it by June. So, how do you even begin? How do you do that? We just know we have a little piece to begin with. We will just do that piece and show them and see how they feel about it. Now we got a little working piece and we came to know a little more. Let's add that to it. When we got to the end and we did the retrospective, I remember very clearly - there was a comment. '*In the beginning we knew nothing and we grew it together and there is no other way we could have done it*'. There is no way we could have started with a big requirements document. There is no way we could have said that we are going to lay down milestones. We had no idea what the milestones were. Even the customer also did not know. The customer wasn't lying. They knew they needed it by June and there would be an upheaval and somebody else would have a solution viable in the market. They were just telling the truth. And that's the case with everything out there now. We know the solutions are going to be there. We just don't know what they are. So unless we grow it together we are not going to have it. And this was in 1994 and I wrote an article in IEEE software article available in my site <http://lindarising.org>.

We even had an experiment with the testing team. There was lot of process in testing those days. Developers used to write software and throw it to the other side of the wall to the testers and there was a lot of blaming. If you find a problem with the software it is your fault. There used to be wars. So when we started doing Scrum with testers, we told them, you



have to be involved from the beginning. You can't just sit over there and wait till the developers are done. It was revolutionary. The entire organization began to see that this was the way to work. They realized that they could think in a more agile fashion and it changed the whole organization and not just the new projects. They believed that this was a better way to collaborate, communicate and deliver. And all began to move in that direction.

How do you convince the customers to buy the agile idea?

When we started, the first group I talked to, was one of those teams that was in trouble. The team liked the idea. But you know what, the testers in those teams would never go with us. They were used to having everything at the end and they did extensive testing. They would never go along with us for these small increments. Then I go and talk to the testers. You have to talk to the testers from the testers point of view. I didn't give the same presentation, I gave to the developers. I told the testers how testing will become better because of Scrum. You have to step in to the shoes of the tester and feel their problems. You have to understand their pain and come up with your idea. When I did that the testers said, Aha, I can see how it can make my life better.

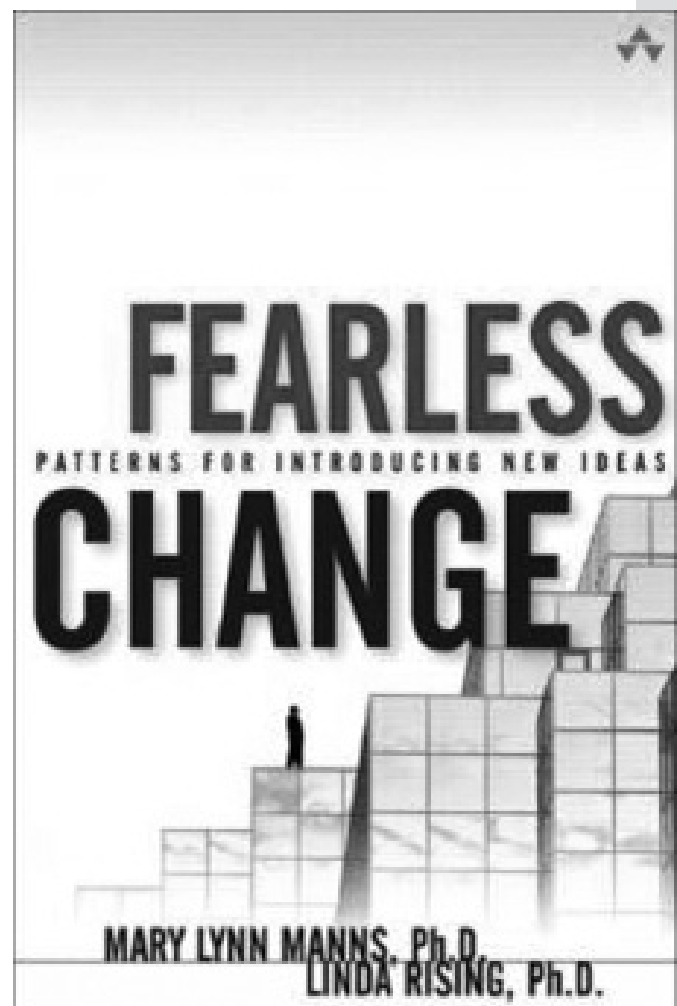
But those people never go along together; the developers, the testers, the business people and everyone say that about the next link in the business chain. They will never go along with us. So you have to tailor the message to address the pain of the group. If you are able to do that, it meets their understanding.

The ideas behind agile are very convincing. The message has to be tailored. It can never be the same for every group. By the time I had gone around the circle and talked to the developers, testers, business people, specifically the sales and marketing people and the customers they all said, 'Oh great. This could be great advantage to us. I will be willing to try it.' Because they knew things were working their way. So when people are in pain they look for better ways. And that's what you need to do. If you just get up say, here is what Scrum is and this is what it does, it wouldn't work. You have to address the pain, which means you have to understand them. *Stephen Covey* wrote a great book called *The Seven Habits of Highly Effective People*. He said if you are going to work with other people, eventually you have walk in their shoes and you need to know how they see the world and it is not always

easy. You have to go talk to them, see what they do, watch, listen and then develop your presentation and that's very difficult. We just say, I'll tell how it works for me and it is enough to convince you. We are just not convinced by logic, but by someone who understands and translates for me. Sales guys understand that, Marketing guys understand that, but that may not really mean anything to me. You have to know me if you have to sell your product and not talk blah... blah...blah about your product. I care about what I want. And those of us who are technical people, we are not used to thinking that way. So if you give a logical explanation of the product it will still not be effective.

And that leads us to understanding the culture. How do you sell Agile to a team with diversified culture?

One of the things to realize is that diversity is good. Diversity has its advantage, because it brings in



different points of view to the table and different ways of doing the work. Many companies in many countries hire people because they are just like one of them. We like people who are like us. What you get from that is unity and a viewpoint that stand in the way of innovation. So the first thing you need to do is embrace innovation and that's your strength.

You need to understand that we don't have to agree upon everything. I know we don't agree, but we should share those points and that's your power and you need to build on it. So this will also enable you to build a better product and communicate with a variety of customers. There is going to be someone in your team who understands a particular variety of customer. If everyone in your team is from same culture, you are going to miss out a customer from different culture, because you don't have an idea about them.

How do you learn to embrace different cultures in a team?

The skill you need to have is listening. We all think listening is not talking. And, what we do when we are not talking and the other person is talking is that we are getting ready. We are preparing our response, preparing our arguments without understanding what the other person is talking. People just don't listen. It takes a lot of practice.

In India, you might be very good at a metaphor about listening. Here you are used to thinking certain people as Gurus. Your idea of going on a pilgrimage to see a Guru is your idea of listening to a wise person. You want to learn everything you can from the Guru. We all need a metaphor and may be this metaphor will work for you. So, when you are talking to a person from different culture from your team, you need to think of that person as your Guru. It's your job to pull out every bit of information from him, so that you can make your mind better. So listening is not about focusing on replying to the person who is talking. It is about pulling the wisdom from the person who is talking. So you need to listen as hard as you can.

I have a friend who is a marriage counselor. When she has a husband and wife who are in conflict, all she says is let me watch when you both talk to each other. She asks them to speak and then stops and asks the husband to tell what his wife just spoke. But, he wouldn't know because he wasn't listening. He was just focusing on his reply. And the same thing would

apply to the wife too. Many conversations are like that. We are always working on our response, because we want to win. So on a diverse team if you don't listen, it becomes a fencing match. So we have to carry the Guru metaphor all the time with us.

If you want to convince someone to use agile, we know that they are going to raise objections and we need to be prepared for that. We always go with the list of items in our head. When he is talking about something you are going to be picking up one of these items and throw it to him thinking that it would be convincing for him. No. It doesn't work at all. The pattern in fearless change says *deep listening is more convincing*. You can just keep listening and it will convince the person, because *listening is a much better communication technique* than using words. I have done that and the first time I thought it wouldn't work and I almost gave up. But I thought about the metaphor and said I am not going to talk to this person at all. I just listened, listened and listened to the person and didn't open my mouth at all. And the person finally said, 'Alright, may be I should give it a try', and all I did was just listen. I listened to him and made him agree. I didn't use logic; I didn't use arguments; I didn't use anything. Most people don't have a clue how powerful this technique is. A person who is being listened is very overwhelmed by this fact and starts opening up to you more and starts thinking as well.

One of the interesting patterns in your Fearless Change book is food. Can you elaborate on that?

It is the **Do Food** pattern. I think it is the most under appreciated pattern. People look at it and say this is silly and ridiculous. We have been eating food together as long as we have been humans. It is a deep human connection we have. I don't know how it works in India. In France say for example, the word friend means *compagnon* in French and it means someone who shares bread with me. Our ancestors, in stone age, only ate food with people who they trusted and that is how it still is. *Sharing food is a sign of trust*.

So here is an experiment that was done to convince people. There are many versions of this experiment. It is about getting a group of people to vote. So there is this control group that doesn't get anything and another group that gets food. There is a presentation made on the issue that we want their support to vote. We give the same presentation, same speech, the same



way to both groups. Later in the study, the group that got food voted in support of the issue and that has been repeated over and over again on multitude of things where you want to convince people. Always happens that the group that gets food is more open and is more likely to support what ever you are talking about. In these experiments, the idea may not be great. You may be asking people to do things that may not be their best. Sometimes it may be about increasing the taxes, with students it may be increasing their fees, so it may not be in their best interest to support it. But the results are still the same. Even if it not a good idea, feeding them will lead them to do it. So, it is not the quality of the idea that is convincing, it is the quality of the food. If you feed them bad food, they will vote against you, even if the idea is good. So, it has to be tasty food. Once I tried to experiment it with healthy food, lot of figs and whole wheat and they hated it. They didn't like the idea either.

Food has always been an enjoyment. I don't know whether you know *Roger Ebert*, a famous movie critic. He had a serious kind of cancer that destroyed his lower jaw. They did all kind of radiation treatment on him and he lost his ability to eat. So he had to be fed with a tube and he could never enjoy his food again. He said you might wonder if I miss the taste of food or drinking my favorite kind of coffee. He said that's not what he missed. He said he missed sitting down with other people and enjoying his food. I realized how deep that connection is when you share food that I cannot participate anymore. So it is a powerful thing that we don't realize.

Very simple, but extremely powerful. We don't realize this in a professional set-up. Don't we?

We have people in our organizations who would just go around the meetings to check if there was food served. In all my talks people have always served good food. We have good nice cookies during the breaks, nice lunch between the talks, and chocolate cakes and ice creams and whatever that makes the participants happy.

We don't realize how important that is and when organizations cut down on food due to budget constraints thinking that it is a luxury and waste of money, I just tell them they are overlooking its power. I personally always use food in retrospectives. Because retrospectives are always unpleasant. Having food



Christopher Alexander
said a pattern can
be implemented in
a million ways, but
the essence remains
constant.



during retrospectives helps us get past our bad times. There are some types of food, which I call *comfort food* that need to be served during these meetings, because they remind us of happy times. So food removed from an organization means, we don't have the comfort and connect anymore. And for cutting cost you are talking about such small amounts. It is a tragic thing for organizations to do that. We need to get that across to the CEOs who make those budgeting decisions. We need to tell them that having food at workplace is an inevitable mark of work culture. So I have experienced this where I was working with a team that had to get a report to the manager every week and that was extremely boring work. Everyone hated getting that report ready. So what I did was one fine day before doing that work and I went to the cafeteria and got some nice little chocolate cookies. And the moment I put those on the table everybody in the team started telling their stories and all of a sudden the report was done. It happened while we were eating and telling stories about the cakes. All the work was done and we finished early. We all walked off saying, why don't we have this everytime.

Did you have a team to research all those patterns in your book or was it all done by you?

Well. A pattern cannot be your idea always. Starting from the Gang of Four patterns, they have always said, we four guys have watched such things in so many places. It was not a single person's idea. But we all have agreed upon these and had validations from various users who have seen the idea. So patterns are a work of a community or a group of people who have seen





and used ideas what ever the domain may be. So if you are going to write a pattern, you have to write to the patterns conference. You have a Shepherd who writes down these patterns and gives a feedback. Then you go to a patterns conference, sit in a small group and discuss the pattern. Only after that it becomes a pattern. For example, the Do Food pattern, I found out in this group that people in Japan don't share food. But they all go out after work and share a beer later. How you implement the Do Food pattern is different in different countries. *Christopher Alexander* said a pattern can be implemented in a million ways, but the essence remains constant.

One last politically incorrect question.

Oh, I love politically incorrect questions. When George Bush II was elected as the President of the US, I went around with a T-Shirt that said, I didn't vote for your father either.

Haha... Agile is all about responding to change and responding to failure. As we grow old it becomes harder. How do you at 72 cope up with change?

It's human nature to resist change. Yes. As we grow old, it is very difficult. But what we do like is the idea to experiment. We may not like it and we may not change, but we are usually open to the idea of small experiments. May be we can try it. It won't be that hard. If we like it we can use it. If we don't, we need not worry. So these are counter forces. On the one hand we resist change, because people want to make big changes, faster changes and that is very hard and may never happen. And we will resist that as much as we can. On the other hand, we as humans have been alive and what kept us going was we are willing to experiment. If our ancestors had not experimented, we wouldn't have come this far. They were all risk takers, but to a limited extent. They were willing to move forward talking little steps and travel all over the globe. So we have to remember that big changes are very painful, but little experiments are always fun. Agile is all about little tiny risks and taking small steps with small learning and small changes. Don't try to push too much and bring about an upheaval. It is true even in the personal level. Agile is not like New Year resolutions like I want to run a marathon and then fail miserably. It's like, may be I want to walk every Tuesday after the dinner and that will work and slowly you extend it further. ■

Pragmatic **P**rogrammer **Q**uotes

“Prolific developers don't always write a lot of code, instead they solve a lot of problems. The two things are not the same.”

- J. Chambers

“Beauty is more important in computing than anywhere else in technology because software is so complicated. Beauty is the ultimate defence against complexity.”

- David Gelernter

EXT JS 5 and Sencha Touch

Workshop and Corporate Training

For the Latest Training on Javascript Toolkits and Mobile

Scan to visit us
on the web



durasoftindia.com

info@durasoftindia.com or 98842-94494

Effective Java with Groovy



Effective Java is probably one of the best books ever written for Java programmers. It presents us effective ways and good practices of using the language. Groovy language provides solutions for many such good practices out of the box. In the second part of the series of articles, I will walk you through code examples that follow these good practices. You will be pleasantly surprised by the simplicity with which Groovy empowers the developers and lets you to focus on solving the problems at hand. I am going to select some recommendations from 'Effective Java (II edition)' in random order and present you the techniques that Groovy offers. I will present a title along with its reference number in the book and discuss it in Groovy.

Obey the general contract when overriding equals (item 8); Always override hashCode when you override equals (Item 9)

Often developers overlook the importance of getting the object equality right. Consider the following code sample.

```
class Book{
    String isbn
    String title
}
def book1 = new Book(isbn: '9788131726594', title:
'Effective Java')
def book2 = new Book(isbn: '9788131726594', title:
'Effective Java')

assert book1 == book2 // fails
```

The assertion in the above code fails indicating *book1* and *book2* are not equal. This happens because the class *Book* will inherit the *equals* implementation from *java.lang.Object*, where two objects are equal only when they point to the same memory location. However while developing applications, we would want *book1* and *book2* objects to be equal. Before we get into solving this problem, let's discover another problem.

```
def book1 = new Book(isbn: '9788131726594', title:
'Effective Java')

def stock = [:]
stock[book1] = 100

def book2 = new Book(isbn: '9788131726594', title:
```

```
'Effective Java')
println stock[book2] // null
println stock // [Book@3c255a5a:100]
```

Here I want to store the stock count for each book in a *map* variable, *stock*. I use the *Book* instance as key and the stock count as value. Later when I want to retrieve the stock, I would get the inputs from the user and construct a *Book* object named *book2*. However the map does not return a value against *book2*, even though it has an entry for the **corresponding ISBN**. The problem is we are again relying on the default implementation of *hashCode()* and a *HashMap* relies on *hashCode()* to get to the bucket where the object is searched using *equals()*. In Java, you would go ahead **overriding equals()** and *hashCode()*. Let's hold on for a moment to study the problems with such implementations. When you write a class, you would ask your favourite IDE to override *equals()* and *hashCode()* methods for you, than writing that boring boilerplate code yourself. Later when you add a property to the class, developers may forget to modify the overridden methods or modifying only one of them. Thus the approach fails to provide a single point of representation of this knowledge, considering both *equals()* and *hashCode()* implementations would rely on the same set of properties. Groovy solves this problem by providing *@EqualsAndHashCode* annotation. The following code snippet shows how to use this annotation.

```
import groovy.transform.EqualsAndHashCode @Eq
ualsAndHashCode(includes='isbn')
class Book{
    String isbn
    String title
```

```

}
def book1 = new Book(isbn: '9788131726594', title:
'Effective Java')
def book2 = new Book(isbn: '9788131726594', title:
'Effective Java')
assert book1 == book2 // passes

def stock = [:] stock[book1] = 100
println stock[book2] // 100

```

You can also consider using the following attributes for customizing the code generation.

- **cache:** For immutable classes, consider setting this to true, so that hash code need not be computed each time it is called
- **callSuper:** Set this to true for sub-classes
- **includeFields:** Set this to true to consider fields in addition to properties

Caution for Grails users

It is highly recommended to apply `@EqualsAndHashCode` annotation on all domain classes. You might be tempted to specify the `id` in the `includes` attribute like `@EqualsAndHashCode(includes='id')`. But note that a domain class instance will get an `id` only after the object is persisted, if the default strategy of `id` is auto generated. Hence have `yourequals()` and `hashCode()` implemented based on the business keys.

```

import groovy.transform.EqualsAndHashCode @Eq
ualsAndHashCode(includes='isbn')
class Book{
    Long id
    String isbn
    String title
}

```

Avoid float and double if exact answers are required (item 48)

Let's consider the following Java code.

```

public static void main( String[] args ) {

    float price = 0.1f;
    float total = 0.0f;
    for(int i=0; i<10; i++){
        total += price;
    }
    System.out.println(total); // 1.0000001
}

```

In any business application, the user would expect the answer to be `1.0` but it turns out to be `1.0000001`, which isn't precise. At this time you might be intrigued to know the impact of changing float to double. If you do that, you should get `0.9999999999999999!`. As the title suggests you should avoid float and double if exact answers are required. If you are a fairly experienced Java developer, you would know that you should use `java.math.BigDecimal` instead.

```

BigDecimal price = new BigDecimal("0.1");
BigDecimal total = BigDecimal.ZERO;
for(int i=0; i<10; i++){
    total = total.add(price);
}
System.out.println(total); // 1.0

```

Here are some common traps that you might fall into, while working with `BigDecimal`.

- While constructing a `BigDecimal` object, you have to pass the numeric value as a `String` object. Since double has already lost precision, passing a double value will not help.
- `BigDecimal` is an immutable class. You might forget to assign the result of `total.add(price)` back to a variable.

Let's find out how Groovy can improve this experience for a developer.

```

def price = 0.1;
def total = 0
10.times{
    total += price
}
println total // 1.0
println price.class // class java.math.BigDecimal

```

In the Groovy version of the code, you need not explicitly specify the type. However the default types have produced the expected result `1.0`. This is because Groovy will assume `BigDecimal` type for any floating point numbers (unless you specify to use something else). Choosing appropriate defaults can impact developer productivity positively and avoid surprising results. Groovy has taken a right step here, as mostly you would use Groovy for writing business applications.

Prefer for-each loops to traditional for loops (Item 46)

Iterating through the elements of a collection is a very common task that we perform in our code. Traditional `for` loop in Java makes developers to write very verbose code by declaring an index variable (with `i` being the popular choice!), increment it by 1 and specify the terminating condition. Java

5 introduced *for-each* loop (*for(:)*), which is a syntactic sugar over *Iterator*. Below is the Groovy version of *for-each* loop.

```
def numbers = [2, 1, 5, 4]
int sum = 0
for(int number in numbers){
    sum += number
}
println sum // 12
```

Java 8 also provides some better alternatives, let's find out what Groovy has to offer. The above code is still not an idiomatic Groovy code. While by embracing *for-each* loop, we got rid off maintaining the index variable, but we still keep modifying the variable *sum*. By using internal iterators provided by Groovy, one can get away from maintaining intermediate states. The following example shows how we can print the elements of a collection one by one using an internal iterator.

```
numbers.each { number ->
    print "$number "
}
// 2 1 5 4
```

To get a better understanding of the value provided by internal iterators, consider the following piece of code, which doubles the elements in a list of numbers.

```
println numbers.collect{ number ->
    number * 2
}
// [4, 2, 10, 8]
```

The following code block shows how to compute the sum of a list of numbers.

```
println numbers.inject(0){sum, number ->
    sum + number
}
// 12
```

Thus by using internal iterators, we avoid maintaining redundant state information, which in turn reduces errors introduced by accidental modification of state. So in the modern era, the advice provided by Josh can be reframed as **prefer internal iterators to external iterators**.

Use function objects to represent strategies (Item 21)

Say you have a list of numbers and you want to perform the following operations.

- Find all even numbers
- Find all odd numbers
- Find numbers divisible by 4

- Find numbers greater than 5

Essentially we want to filter numbers based on different criteria. We can use strategy pattern here to separate 'how to filter' from the actual 'filtering criteria'. Approaching this problem with an orthodox OOP style, one would create a single method interface called *Filter* and provide implementations for each criterion, resulting in an explosion of classes. Often developers name these implementation like *Filter01*, *Filter02* etc. And this may lead to duplication of code.

In Groovy you can use closures and pass closures as arguments to a function.

```
def numbers = [1, 3, 4, 8, 16, 9]
def odd = { number -> number % 2 != 0 }
def even = { number -> number % 2 == 0 }
def divisibleBy4 = { number -> number % 4 == 0 }
def greaterThan5 = {number -> number > 5 }
```

```
assert numbers.findAll(even) == [4, 8, 16]
assert numbers.findAll(odd) == [1, 3, 9]
assert numbers.findAll(divisibleBy4) == [4, 8, 16]
assert numbers.findAll(greaterThan5) == [8, 16, 9]
```

As you can observe, we have a concise code, which is easy to read and maintain.

References

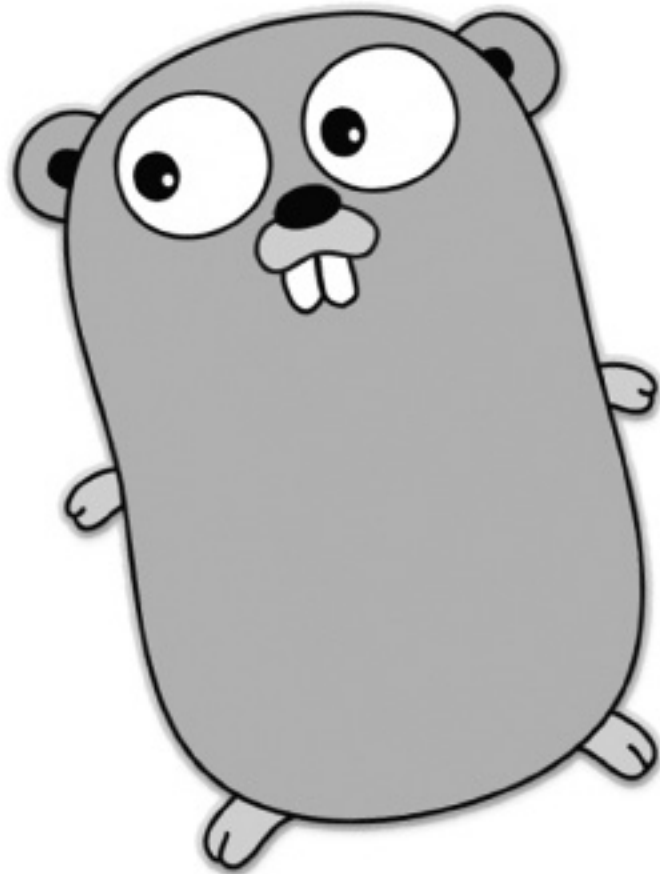
- Effective Java, Second Edition



Naresha works as Chief Technologist at Channel Bridge Software Labs and has more than eight years of experience. He is passionate about learning new technologies and most of his current works are on Groovy and JavaScript. Naresha is also an organizer of the Bangalore Groovy Grails Meetup.

Email:naresha.k@gmail.com
Twitter:@naresha_k

Let's go
with



= Go

*Here's another language called **Go**. Every programming language is a mechanism used to communicate with the computer. Come to think of it, computers are the same, so why do we need so many languages? Given a certain task, it can be written as an instruction to the computer using any programming language. Why are new programming languages being developed then and why should we care to learn a new one? And does a new language like Go try to solve anything that's not possible using other languages we already have? Let's find that out.*

Where does Go fit?

Some languages are more specialised for certain tasks such as **R** for statistics, **Erlang** for concurrency, **C** for low-level system programming and so on. As computers and humans evolve, we invent ways to solve problems elegantly while making full use of available system resources. In today's world while we may have reached a limit on how fast a single CPUs can be, modern CPUs achieve scale by having many cores, server CPUs can easily have as many as sixteen cores. Also, at the same time memory and storage get cheaper with each passing day. What is still costly though is the programmer's time. Modern programming languages such as **Ruby** and **Python** go a long way in writing complex programs quickly and elegantly. However, Ruby and Python lack speed and support for concurrency. **Scala** solves the last two problems, however, one can find the learning curve to be quite steep and reading someone else's code a bit difficult.

We can sense a void in the programming world. We are always looking for a programming language that is fast, simple, has support for concurrency and supports modern paradigms like first-class functions. Enter Go. Go programming language or **Golang** is a programming language backed by Google. It is designed to be a simple, statically-typed language, with garbage collection and built with concurrency in mind. If a young programmer were to ask me why he/she should learn Go, I would simply say that it is 'easy to learn', it is 'fast', and the language has in-built support to be 'highly concurrent' while avoiding common issues associated with concurrency.

Let's look at some of the features in detail.

- **Simple:** Go is easy to learn. Period. The main language features can easily be learned by doing the 'Official Go Tour' on <https://tour.golang.org/> in a couple of days. With

some programming experience and a little effort one can start writing Go programs quickly which is not the case with other programming languages like Scala or Clojure that have quite a steep learning curve.

- **Type safety with type inference:** While type safety is nice, it can be verbose (I am looking at you, Java). Like Scala, Go provides type safety and infers type intelligently which not only helps you write large programs safely, but also does not get in your way while programming. The language documentation categorically states that Go is an attempt to combine the ease of programming of an interpreted and dynamically-typed language with the efficiency and safety of a statically typed, compiled language.
- **Garbage Collection:** While garbage collection can be slightly expensive, it saves a lot of programmer time. Another important point is that a large part of the difficulty in concurrent and multi-threaded programming is memory management. Automatic garbage collection makes concurrent code far easier to write. Of course, implementing garbage collection in a concurrent environment is itself a challenge; but meeting it once rather than in every program helps everyone. In the current version of Go, garbage collection uses the simple 'mark-and-sweep' pattern (used in older Ruby versions as well), while the future versions plan to introduce more efficient garbage collection algorithms.
- **Binaries for all major platforms:** One of the most annoying problems is software distribution. If you want to run a Java application you need the Java Runtime Environment. Similarly, all Ruby programs need the Ruby interpreter to be installed before you can do anything. Not to mention the package and dependency managers. Also, if you need to install something on the

user's machine (like a script or a desktop app) things get even murkier. Go has no such problems – it generates a single binary file for all major platforms. Just transfer it and run it. That's it.

- **Concurrency support:** As mentioned earlier, multi-core CPUs are the norm now. To utilize their processing power available, the programming language needs to support some form of concurrency. Dealing with low-level threads is cumbersome, highly error-prone and hard to debug. There needs to be some abstraction that helps deal with concurrency. Scala provides this with the **Actor model** and Go does it using **Channels**. Go's philosophy is: "Don't communicate by sharing memory; share memory by communicating." Channels allow you to pass references to data structures between **goroutines**. If you consider this as passing around ownership of the data (the ability to read and write it), they become a powerful and expressive synchronization mechanism. What's more? Channels are a core language feature and no external library is needed to write highly concurrent programs.

One of the best things about Go is that the standard library is super rich and takes care of most common needs. Things like 'HTTP server', 'cryptographic utilities', 'compression', 'unicode' are all baked into the standard library. For most common tasks, you need not import any external library. Although the library ecosystem is continually growing, for a language introduced in 2009, you can find all sorts of libraries and frameworks on Github. There are full-fledged web frameworks like **Revel** and **Beego** and there are desktop application development libraries like <https://github.com/andlabs/ui>. The language also releases all future development plans to the public, the code now lives on Github, there are releases every six months and there seems to be a clear direction as the language is progressing.

Growing adoption by leading technology companies

New programming languages face challenges when it comes to adoption. Go, on the other hand, has had no such problems. Since its introduction around five years ago, Go is used in production by Google, Dropbox, Soundcloud, BBC and some more. The next big thing in Linux container management, **Docker** is written in Go. This ensures that the language is here to stay, there are and will be jobs and the early adopters will, of course, have a slight advantage.

Alright, enough talking. Let us start writing some code.

First Go Program

Here's our first, 'Hello World' program in Go.

```
package main //1

import ( //2
    "fmt"
    "os"
)

func main() { //3
    fmt.Println("Hello " + os.Args[1])
}
```

With Go installed, this can be run like this:

```
go run main.go 'World!'
```

Let me quickly run through the code by explaining the comments numbered from 1 to 3.

1. Every program in Go is made up of **packages**, the main package is the entry point from where the program is run.
2. The import statement is used to import external packages. Both **fmt** and **os** packages are part of the standard library. We don't need to do anything special to load them.
3. The main function is run, printing **Hello** with any command line argument is passed in.

Using the built-in os package

Let us write another program that uses the Operating System (os) package.

```
package main //1

import ( //2
    "fmt"
    "os"
    "os/exec"
)

func main() {
    query := os.Args[1] //3
    out, err := exec.Command("find", ".", "-iname",
query).Output() //4
    if err != nil { //5
        panic("Error!")
    }
    fmt.Printf("File: %s\n", out)
}
```

Sections 1 and 2 are similar to the last program we wrote. In section 3, we use **Type Inference**. The variable 'query' has no declared type, but it is inferred from the return value of 'os.Args' function. What is also interesting is that all functions in a package that start with a capital letter are exported automatically (which I think is quite neat). In section 4, we use the unix 'find' command and store the output in an 'out' variable. We exit or 'panic' out in case the 'find' command returns an error. And we finally print the result.

If we run the program using **go build**, we will get an executable. Let's copy the executable to '\$HOME/bin' and rename it as 'lookup' and voila! We have a lookup command available to us.

A simple web server in Go

This example is taken from the Go wiki. Let's create a simple web server.

```
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter,
    r *http.Request) {
    fmt.Fprintf(w, "Hi there, I love %s!",
r.URL.Path[1:])
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

The 'main' function begins with a call to 'http.HandleFunc', which tells the 'http' package to handle all requests to the web root ("/") with handler.

It then calls 'http.ListenAndServe', specifying that it should listen on port 8080 on any interface (":8080"). (Don't worry about its second parameter, 'nil', for now.) This function will block until the program is terminated. The function handler is of the type 'http.HandlerFunc'. It takes an 'http.ResponseWriter' and an 'http.Request' as its arguments. An 'http.ResponseWriter' value assembles the HTTP server's response; by writing to it, we send data to the HTTP client.

The 'http.Request' is a data structure that represents the client HTTP request. 'r.URL.Path[1:]' is the path component of the request URL. The trailing '[1:]' slices the leading "/" from the path name. We can also notice that 'http.Request' is passed in as a pointer. This is because normally in Go arguments

are passed by values and copied to the function. When we declare a parameter as '(r *http.Request)' we are saying that the type '*r' is a pointer to a value of 'http.Request' stored somewhere.

If you run this program and access the URL, <http://localhost:8080/monkeys>, the program would present a page containing:

Hi there, I love monkeys!

That's it! With a few lines of code, you have an HTTP server running.

Concurrency

Let us see a simple working example on concurrency. Here is a simple problem where we want to run multiple tasks parallelly. Each task takes a random amount of time, so we run all of them in parallel and wait for all to be completed. The example is simplified but the tasks in real life can be making a HTTP request or a remote API call. Here's the code to do that.

```
package main

import (
    "fmt"
    "time"
    "math/rand"
)

func main() {
    rand.Seed(time.Now().UTC().UnixNano()) //
put in a varying seed
    tasks := []string{"a", "b", "c", "d", "e",
"f"}
    size := len(tasks)
    ch := make(chan string, size)
    for i := 0; i < size; i++ {
        go doTheTask(tasks[i], ch) //spawn a
goroutine
    }
    for i := 0; i < size; i++ {
        result := <- ch
        fmt.Println(result)
    }
    func doTheTask(task string, ch chan string) {
        time.Sleep(time.Duration(rand.Intn(1000))
* time.Millisecond)
        result := "Done with task: " +
task
        ch <- result
    }
```


With Go, the solution to this problem is simple and elegant. Just adding a keyword `go` before a function call, runs it in a separate co-routine. We also pass the channel where the result is published. Since there are six co-routines/tasks, we listen for six responses on the channel and print them as we get them. As you can guess, each 'run' of this program will print a different output. Goroutines and channels allow us to write clean and quick concurrent code.

Final thoughts

As we have seen so far, Go is extremely useful and easy to pick up. If I were to think of a few downsides, it would be that the language has no 'official' package manager right now. Dependency Management can right now be solved by **GPM** and **GVP** (<https://github.com/pote/gpm>), but the community is divided and there is no single way to do it. Also, since the language is so new, the third party packages have to be used with caution. Finally, the Go mechanism of dealing with errors (as seen in example 2 above) can be a bit cumbersome, there are a lot of debates on this at the moment.

However, if you are looking to build powerful command line utilities or simple JSON spewing web servers that can handle thousands of requests per second (while consuming very little system resources), you cannot go wrong with Go. Also, with time the language can only improve, it was first released in November 2009 but has gained tremendous traction in a few years. So, keep your eyes open and learn some Go!



Rocky is a software developer with more than twelve years of experience in software design and programming. He enjoys coding in Ruby, JavaScript, Java and Go. He loves working on open source projects and tinkers with technology in his free time. He is currently working as a software developer for Crealytics GmbH. He has been a speaker at AgileNCR 2010, Agile Tours 2010, IndicThreads Conference 2011/12 and Ruby Conf India 2012/13. His blogs and videos can be found at <http://rockyj.in>.

“Bad programmers have all the answers.

Good testers have all the questions.”

- Gil Zilberfeld

“Don’t document bad code — rewrite it.”

- Kernighan and Plauger

Pragmatic
Programmer
Quotes

The
Pragmatic
Programmers

Functional Programming in Java

Harnessing the Power of
Java 8 Lambda Expressions



Venkat Subramaniam

Foreword by Brian Goetz

Edited by Jacquelyn Carter



GOPHERCON INDIA 2015

FEB 19 – 21, 2015, BANGALORE, INDIA

The first ever Go conference in India

Go is an open source project developed by team at Google and many contributors from the open source community. This conference is brought to you by the Go language community in India.

Venue:

Hotel Royal Orchid

No. 1, Golf Avenue,
Adjoining KGA Golf Course,
HAL Airport Road,
Bengaluru 560 008, India
<http://www.gophercon.in>

NULLCON
www.nullcon.net
the neXt security thing!

NULLCON 2015

FEB 4-7, 2015, GOA, INDIA

Nullcon was founded in 2010 with the idea of providing an integrated platform for exchanging information on the latest attack vectors, zero day vulnerabilities and unknown threats, Our motto - "The neXt security thing!" drives the objective of the conference i.e. to discuss and showcase the future of information security and the next-generation of offensive and defensive security technology.

Venue:

The Bogmallo Beach Resort,
Goa, India

Website: <http://nullcon.net/website/>

WHEN? 23 - 30 March 2015

WHERE? Chancery Pavilion, Bangalore

Agile India is India's premier Agile conferences with focus on Scaling Agile Adoption and Scaling Lifecycle.



Venue:

The Chancery Pavilion Hotel Bangalore
#135, Residency Road, Bangalore - 560 025, India
<http://2015.agileindia.org>

Advertise your company in Healthy Code

Back Cover – Rs. 20,000 /-

Inside Front and Back Covers – Rs. 15,000 /-

Full Page – Rs. 12,000 /-

Half Page – Rs. 7000 /-

Call us today @ 98842 94494



Maximum Value. Maximum Reach.



Skill Test

Know your Java 8

*Hello readers! Enjoy this quiz on Java 8. We're sure you'll have fun.
You can check out the answers in the end.*

Question 1

Does the following interface definition qualify to be a Functional interface?

```
public interface Math{  
    int add(int a,int b);  
    int diff(int a,int b);  
}
```

.....

Question 2

What are Lambda expressions?

.....

Question 3

What is the Lambda equivalent of the following code?

```
interface Adder{  
    int sum(int a,int b);  
}
```

```
}  
...  
Adder adder = new Adder(){  
    public int sum(int a,int b){  
        return a + b;  
    }  
}  
System.out.println(adder.sum(12,13));  
.....
```

Question 4

```
interface Util{  
    String doSomething(String str);  
}
```

Using the *Util* interface you want to return the given string, say "Skill Test" in *uppercase*, and *lowercase*. How do you write that using *Method Inference*?

.....

Question 5

```
interface Util{
    String doSomething(String str);
}
```

Using the *Util* interface you want to return the given string, say "Skill Test" in *uppercase*, and *lowercase*. How do you write that using *Lambda Expression*?

Question 6

What's the output of the following code?

```
Stream.of(1, 2, 3, 4, 5)
    .filter(i -> {
        return i%2 == 0;
    })
    .forEach(System.out::println);
```

Question 7

What's the output of the following Stream code?

```
Stream.of(1, 2, 3, 4, 5)
    .filter(i -> {
        System.out.println("Filter: " + i);

        return true;
    })
    .forEach(s->System.out.println("ForEach: " + i));
```

Answers

- **Question 1:** No. A Functional interface in Java 8, is an interface that has only one method that needs to be implemented.
- **Question 2:** Lambda expressions are shorter ways of writing method implementations. They are smaller versions of anonymous inner classes. Lambdas do a little more than generating anonymous inner classes.
- **Question 3:** The Lambda version of the implementation of *Adder* interface is:

```
Adder adder = (a,b) -> { return a+b; };
```

- **Question 4:** The two implementations of

the *Util* interface using *method inference* are:

```
Util util1 = String::toUpperCase;
System.out.println (util1.doSomething("Skill Test"));
```

```
Util util2 = String::toLowerCase;
System.out.println (util2.doSomething("Skill Test"));
```

- **Question 5:** The two implementations of the *Util* interface using lambdas are:

```
Util util1 = s->{ return s.toUpperCase(); };
System.out.println(util1.doSomething("Skill Test"));
```

```
Util util2 = s->{ return s.toLowerCase(); };
System.out.println(util2.doSomething("Skill Test"));
```

- **Question 6:** The code using *Stream* API, filters out all the even numbers and prints them.

2
4

- **Question 7:** The code using *Stream* API, runs vertically and not horizontally as we think. The output of the code is:

Filter:	1
ForEach:	1
Filter:	2
ForEach:	2
Filter:	3
ForEach:	3
Filter:	4
ForEach:	4
Filter:	5
ForEach:	5

References

<http://www.oracle.com/technetwork/articles/java/architect-lambdas-part1-2080972.html>

<http://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>

Coming up in the March 2015 issue

PhantomJS:

**A programmable head less browser
– Shani Mahadeva**

PhantomJS is an attempt to automate browser tasks. It is a JavaScript framework developed over QtWebKit that makes use of WebKit content engine. It provides a JavaScript API that can be used to control web browser. However it is “Headless” in the sense that it doesn’t display the content on the screen.

**Interview with
Dr. Richard Stallman**

Dr. Richard Stallman is a software freedom activist and known for launching the GNU project and writing several of the GNU softwares such as GNU EMACS, GNU Compiler and GNU General Public License.

**The Unsung DOM APIs
- Hemanth**

This article will introduce you to the unsung DOM APIs, most of which you would not have heard of!

Are you subscribed
to India's leading
programming
magazine?

Visit:

<http://healthycodemagazine.com>

EMC²

NULLCON
www.nullcon.net
the neXt security thing!



**ULTIMATE DEFENDER RECEIVES CASH PRIZE OF
INR 5LACS**

DATE: 6TH FEB 2015

VENUE: NULLCON 2015 @ BOGMALLO BEACH RESORT, GOA

<http://nullcon.net>

NULLCON INTERNATIONAL SECURITY CONFERENCE GOA 2015