From SOLID (Principles) to LIQUID (Streams) – An Exploration of Programming Models

Praseed Pai K.T.
@ InApp,Trivandrum

About the Presenter

- A Seasoned Software Engineering Professional with more than twenty five years of Exposure
- Author of Two books on Computer Programming
- Explorer in “Philosophical Tools for Software Engineering” (Has Presented on it, Written one university accredited paper, Designed a Pattern based on Advaita Vedanta to transition from OOP to FRP)
- An Expert level professional in Cross Cultural Encounters (How to deal with a Russian/Eastern European? , Working with Racial stereotypes like Jews / Chinese / Latin Americans)
- A Critique of Digital Technology Fads (Programmers will be better off , if they stick to Programming. Do not run after so called AI/ML, BlockChain etc) - “Plumbing is preferred over Painting!”
- I also help Programmers eliminate their “Math-Phobia”

Who is an Architect?

Architect (n) – Any person who has “fooled” around in the Software Industry for sizeable time (ever shrinking span) who is past his prime,as a Programmer Or Engineer, Systematically moved up in the hierarchy to obey “Peter Principle”.

The Goal of this Presentation

- SOLID Principles as a Verification Tool
- Better API design through Successive Refactoring for Procedural, Object/Component, Hierarchical, Iterative and Reactive model of Programming
- Emphasis on Programming models like OOP, FP , Generic Programming (GP) and Rx Programming
- Will help one to appreciate the work which has gone into creating the framework one uses (To become a Literate Consumer of it!)

Four Types of Parametrization and their associated LINGO

- Data Parametrization (Transaction Script, Prcoedural)
- Variable Parametrization (Class , OOP)
- Type Parametrization (Generics, OOP/GP)
- Behavior Parametrization (Lambda, GP/FP)
- https://github.com/praseedpai/WhetYourApettite

Data Parametrization (Step #1)

```
static class Program{
        static void Main(string[] args){
         //---- Check whether there is any Command Line argument
        if ( args.Length == 0 ) {  Console.WriteLine("No Command Line ARguments");    return; }
       //------ Allocate Array
        int [] arr = new int[args.Length];
        for( int i=0; i< arr.Length ; ++i )
                arr[i] = Convert.ToInt32(args[i]);
             //---------- A Bubble Sort Routine
             int n = arr.Length;
             for(int i = 0; i<n; ++i)
               for (int j = 0; j < n-i-1; j++)
                  if (arr[j]>arr[j + 1]){    int temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;}
            //------------------- Spit the Output
            foreach( var n2 in arr )  { Console.WriteLine(n2); }
    } // Main
} // PRogram
```

Variable Parametrization

```
interface IComparitorStrategy {  int Execute(Object a, Object b); }
class IntComparitor : IComparitorStrategy {
       public int Execute(Object a, Object b) { int  pa  = (int)a ;   int pb  =(int) b ; return pa > pb ? 1 : (pb > pa ) ? -1 : 0; }
}
static class Program{
        private static void BSort(this int[] arr, IComparitorStrategy test) {
           int n = arr.Length;
             for(int i = 0; i<n; ++i)
               for (int j = 0; j < n-i-1; j++)
                  if (test.Execute(arr[j],arr[j + 1]) > 0) {  int  temp = arr[j]; arr[j] = arr[j + 1];  arr[j + 1] = temp;  }
       }

     static void Main(string[] args){
        //---- Check whether there is any Command Line argument
       if ( args.Length == 0 ) {  Console.WriteLine("No Command Line ARguments");    return; }
      //------ Allocate Array
       int [] arr = new int[args.Length];
       for( int i=0; i< arr.Length ; ++i  ) {  arr[i] = Convert.ToInt32(args[i]); }
        //---------- A Bubble Sort Routine
        arr.BSort(new IntComparitor ());
        //-------------------- Spit the Output
         foreach( var n2 in arr )  { Console.WriteLine(n2); }
    } // Main
} // PRogram
```

Type Parametrization

```
interface IComparitorStrategy<T> {  int Execute(T a, T b);}
class IntComparitor : IComparitorStrategy<int> {
        public int Execute(int a, int b) {  return a > b ? 1 : (b > a ) ? -1 : 0;  }
}
class DoubleComparitor : IComparitorStrategy<double>{
        public int Execute(double a, double b){    return a > b ? 1 : (b > a) ? -1 : 0;}
}
 static class Program{
        private static void BSort<T>(this T[] arr, IComparitorStrategy<T> test) where T : struct{
            int n = arr.Length;
              for(int i = 0; i<n; ++i)
                for (int j = 0; j < n-i-1; j++)
                  if (test.Execute(arr[j],arr[j + 1]) > 0)   {     T temp = arr[j]; arr[j] = arr[j + 1]; arr[j + 1] = temp;}
        }
       static void Main(string[] args) {
            if ( args.Length == 0 )    {  Console.WriteLine("No Command Line ARguments");  return; }
            int [] arr = new int[args.Length];
            for( int i=0; i< arr.Length ; ++i )
                 arr[i] = Convert.ToInt32(args[i]);
            arr.BSort(new IntComparitor ());
            foreach( var n2 in arr )  { Console.WriteLine(n2); }
        }
    }
```

Behavior Parametrization

```
static class Program{
      private static void BSort2<T>(this T[] arr, Func<T,T,int> test) where T : struct {
          int n = arr.Length;
          for (int i = 0; i < n; ++i)
              for (int j = 0; j < n - i - 1; j++)
                  if (test(arr[j], arr[j + 1]) > 0)  {T temp = arr[j]; arr[j] = arr[j + 1];  arr[j + 1] = temp;}
      }
      static void Main(string[] args){
          if ( args.Length == 0 ){Console.WriteLine("No Command Line ARguments");  return;  }
          int [] arr = new int[args.Length];
          for( int i=0; i< arr.Length ; ++i ) {arr[i] = Convert.ToInt32(args[i]); }
          Func<int ,int ,int> fn = (int a, int b ) => { return (a > b) ? 1 : -1; };
          arr.BSort2(fn);
          foreach( var n2 in arr ) {  Console.WriteLine(n2);}
      } //Main
 }//Program
```

OOP – A quick introduction

- A class has got a structure (instance variable and class level variables and behaviors (instance methods and class level methods) to provide some functionality
- For large programs , Structure and Behavior needs to be separated for SOC
- Abstraction, Encapsulation , Polymorphism, Inheritance
- Interface and abstract classes

SOLID Principles

Sr. no	Principles	Description
1	**Single Responsibility Principle (SRP)**	A class should have only one responsibility. If it is doing more than one unrelated thing, we need to split the class.
2	**Open Close Principle (OCP)**	A class should be open for extension, closed for modification.
3	**Liskov Substitution Principle (LSP)**	Named after Barbara Liskov, a Turing Award laureate, who postulated that a sub-class (derived class) could substitute any super class (base class) references without affecting the functionality. Even though it looks like stating the obvious, most implementations have quirks that violate this principle.
4	**Interface Segregation Principle (ISP)**	It is more desirable to have multiple interfaces for a class (such classes can also be called components) than having one Uber interface that forces implementation of all methods (both relevant and non-relevant to the solution context).
5	**Dependency Inversion (DI)**	This is a principle which is very useful for framework design. In the case of frameworks, the client code will be invoked by server code, as opposed to the usual process of the client invoking the server. The main principle here is that abstraction should not depend upon details; rather, details should depend upon abstraction. This is also called the **Hollywood** principle (Do not call us, we will call you back).

The Design Pattern "Movement"

- GOF Design Patterns
  - Structure , Behavioural and Creational Patterns
- "POSA" Catalogue
  - Patterns of Software Architecture
- DDD Pattern Catalogue
  - Domain Driven Design
- "POEAA" Catalog
  - Patterns of Enterprise Application Architecture
- Enterprise Architecture and the MDA
  - Archetype Pattern
- Enterprise Integration Patterns
  - Apache CAML is based on this book
- J2EE Design Patterns
  - A Pattern Catalogue for Web Application by Deepak Alur et al.

Command Pattern

```
interface Command{ public boolean Execute();}
abstract class BaseCommand implements Command{
        public boolean Execute() {return false;}
}
class MoveCommand extends BaseCommand{
            public MoveCommand( int dx , int dy ) {}
            public boolean Execute() { System.out.println("MOVE ... "); return true;}
}
class RotateCommand extends BaseCommand{
            public RotateCommand( double angle ) {}
            public boolean Execute() {System.out.println("Rotate ");return true;}
}
class ScaleCommand extends BaseCommand{
                public ScaleCommand( double sx , double sy ) {}
            public boolean Execute() { System.out.println("Scale "); return true;}
}
class GraphicsDB{
            public ArrayList<Command> cmds = new ArrayList<Command>();
                 public GraphicsDB() { SavePoint(); }
            public boolean SavePoint() { cmds.clear(); return true;}
            public boolean RestorePoint() { return true; }
            public boolean SendCommand( Command cmd ) { cmds.add(cmd); cmd.Execute(); return true;}
            public boolean UndoLastChange() {
                        int size = cmds.size(); RestorePoint();
                        for(Command c : cmds ){ c.Execute();}
                        return true;
            }
}
public class CommandTest {
            public static void main( String [] args ) { System.out.println("Command Test...");}
}
```

Composite/Visitor (in a page!)

```
//------ A Simple implementation of Visitor Pattern
interface CarElementVisitor {
    void visit(Wheel wheel);
    void visit(Engine engine);
    void visit(Body body);
    void visit(Car car);
}
interface CarElement {
    void accept(CarElementVisitor visitor);     // CarElements have to provide accept().
}

class Wheel implements CarElement {
    private String name;
    public Wheel(String name) { this.name = name;  }
    public String getName() { return this.name;}
    public void accept(CarElementVisitor visitor) {visitor.visit(this); }
}
class Engine implements CarElement {
    public void accept(CarElementVisitor visitor) {visitor.visit(this); }
}
class Body implements CarElement {
    public void accept(CarElementVisitor visitor) {  visitor.visit(this); }
}
class Car implements CarElement{
    CarElement[] elements;
    public CarElement[] getElements() { return elements.clone(); } // Return a copy of the array of references.
    public Car() {
        this.elements = new CarElement[]  { new Wheel("front left"), new Wheel("front right"),
                                            new Wheel("back left") , new Wheel("back right"),
                                            new Body(), new Engine()
                                                    }|
   }
    public void accept(CarElementVisitor visitor) {
                for(CarElement element : this.getElements()) {  element.accept(visitor);}
               visitor.visit(this);
    }
}
```

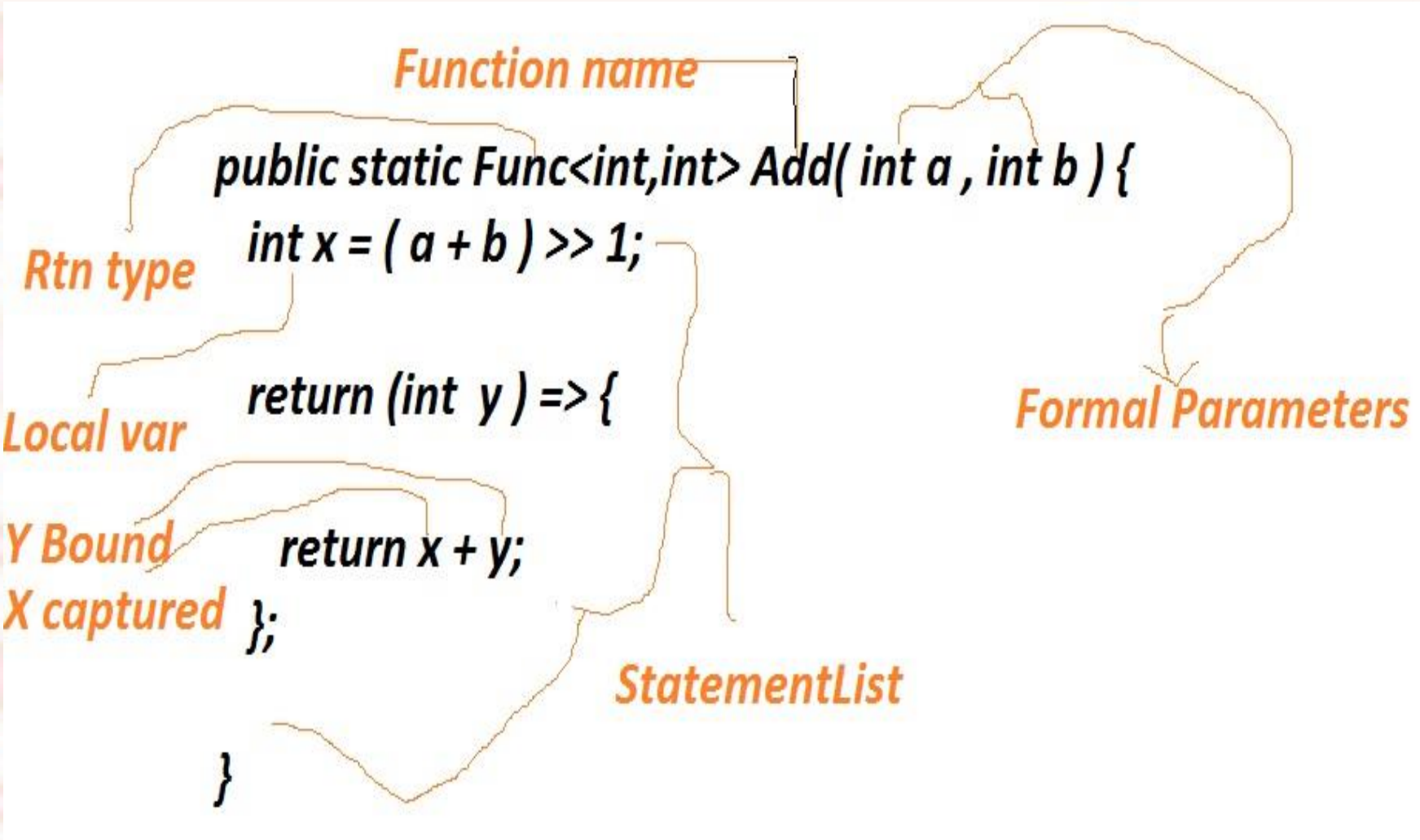
```
class CarElementPrintVisitor implements CarElementVisitor {
    public void visit(Wheel wheel) {  System.out.println("Visiting "+ wheel.getName() + "
wheel"); }
    public void visit(Engine engine) { System.out.println("Visiting engine");}
    public void visit(Body body) { System.out.println("Visiting body");}
    public void visit(Car car) {      System.out.println("Visiting car");}
}
class CarElementDoVisitor implements CarElementVisitor {
    public void visit(Wheel wheel) { System.out.println("Kicking my "+ wheel.getName() + "
wheel"); }
    public void visit(Engine engine) {  System.out.println("Starting my engine");}
    public void visit(Body body) { System.out.println("Moving my body");}
    public void visit(Car car) {  System.out.println("Starting my car"); }
}

public class VisitorTest {
    static public void main(String[] args){
        Car car = new Car(); car.accept(new CarElementPrintVisitor());  car.accept(new
CarElementDoVisitor());
    }
}
```

Generic Programming

- Parametrized Types
- Algorithms are the central citizen of GP
- Highly useful for creating Type Safe Collections
- In C/C++, Generics are called Templates
- C#, Java and C++ instantiates templates differently
- Type Erasure in Java/JVM
- Dynamic Type Synthesis by CLR
- Compile Time Code Factory by C++

FP – Prologue


Function name
public static Func<int,int> Add(int a , int b) {
int x = (a + b) >> 1;
Rtn type
return (int y) => {
Local var
Formal Parameters
Y Bound
return x + y;
X captured
};
StatementList
}

FP – A quick introduction

- LISP is one of the oldest functional programming language
- ML is the first typed functional programming language
- The Lambda Calculus and FP
- C# - an Object/Functional Programming language
- FP is Programming with Pure Functions
- Immutability , Side effects, Composition, Map/Reduce, Behavioral Parametrization , Variable Capture, Referential Transparency, Currying etc

Functional Composition

```
function Compose(a , b ) { return function(x) { return a(b(x)); }}

function sqr( a ) { return a*a; }
function cub( b ) { return b*b*b; }

var c = Compose(sqr,cub);
console.log( c(10) );
```

Currying – a Non Trivial Example

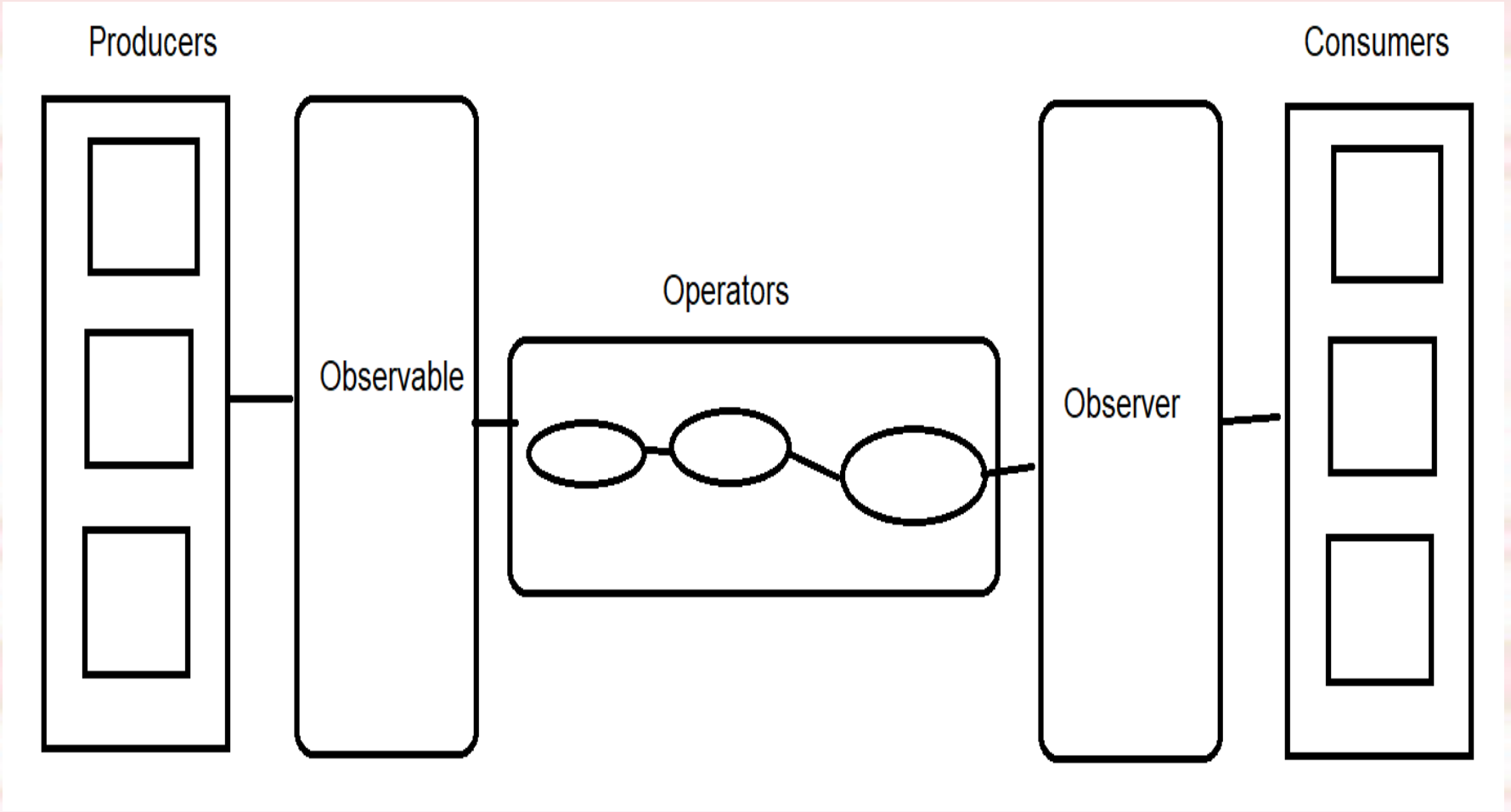
```
Function.prototype.crockfordCurry = function () {
    "use strict";
    var slice = Array.prototype.slice,  args = slice.apply(arguments), that = this;
    return function () {return that.apply(null, args.concat(slice.apply(arguments)));};
};
function add(x) {
    "use strict";
    return function (y) {return x + y;};
}
 function multiply(x) {
    "use strict";
    return function (y) { return x * y; };
}
Function.prototype.haskellCurry = function () {
    "use strict";
    var slice = Array.prototype.slice, that = this;
    return function () {
        var jsEval,  jsApply, fn, args = slice.apply(arguments),result = args[0];
        jsEval = function () {
            args = slice.apply(arguments);  fn = that.call(null, result);
            jsApply.apply(null, args);
        };
        jsApply = function () {
            args = slice.apply(arguments);  result = fn.call(null, args[0]);
            if (args.length > 1) {  jsEval.apply(null, args.slice(1)); }
        };
        jsEval.apply(null, args.slice(1));
        return result;
    };
};
```

```
//Creating the curried functions
var adder2 = add.haskellCurry(),
    multiplier2 = multiply.haskellCurry();
//Finding the sum of 1, 2 & 3 with the curried function
console.log(adder2(1, 2, 3));
//Finding the product of 1, 2, 3, 4, 5 & 6 with the curried function
console.log(multiplier2(1, 2, 3, 4, 5, 6));
//Creating the curried functions
var adder = add.crockfordCurry();
var multiplier = multiply.crockfordCurry();
//Finding the sum of 1,2 & 3 with the curried function
console.log(adder(adder(1)(2))(3));
//Finding the product of 1,2 & 3 with the curried function
console.log(multiplier(multiplier(1)(2))(3));
```

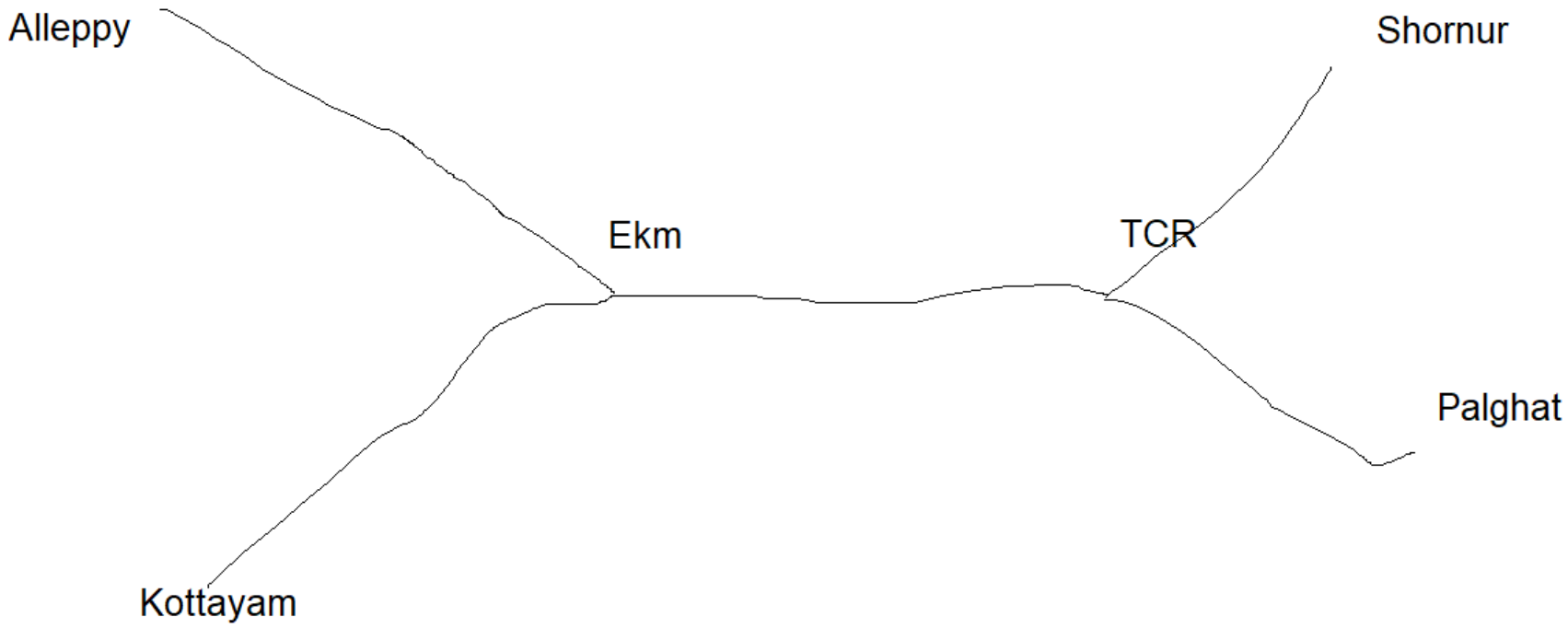
Rx Programming Model

- FRP (Functional Reactive Programming) was pioneered by the Haskell Community
- Microsoft released Rx.net and RxJs
- NetFlix ported Rx.net to Java (RxJava)
- Rx libraries are available for .NET, Java, JS, Python, C++ etc, to name a few

Rx – The Big Picture



Rx – What is happening?



Rx Simple Code Snippets

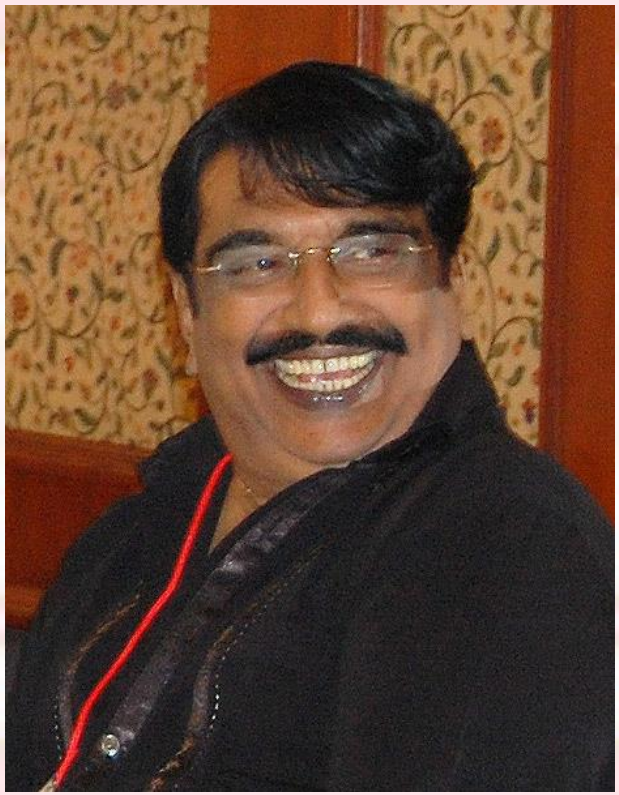
```
Observable<String> observable = Observable.just("how", "to", "do", "in", "java");
 //consumer
Consumer<? super String> consumer = System.out::println;
 //Attaching producer to consumer
observable.subscribe(consumer);
```

```
static void Main() {
        var mylabel = new Label();
        var myform = new Form { Controls = { mylabel } };
        IObservable<EventPattern<MouseEventArgs>> mousemove =
            Observable. FromEventPattern<MouseEventArgs>(myform, "MouseMove");
         mousemove.Subscribe( (evt)=>{mylabel.Text = evt.EventArgs.X.ToString();}, ()=>{});
        Application.Run(myform);
}
```

Case Study #1

- A Directory Traversal Engine (Java)
- A Procedural API for Directory Traversal
- A OOP/Component Model for Directory Traversal
- Directory Traversal using Composite/Visitor
- Directory Traversal using Iterator
- Directory Traversal using Streams
- Directory Traversal using RxJava

Comparitive Study of Programming Languages



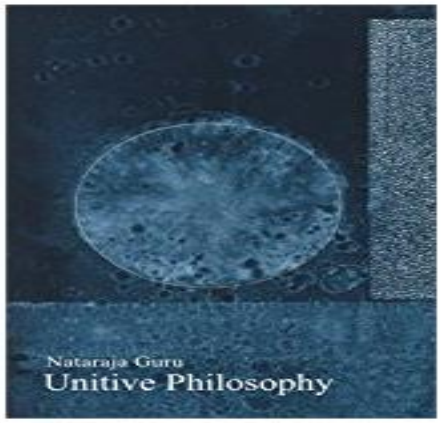
In India, it is like this, I do not know how it is in Punjab?
 - Punjabi House

OOP – What is the Essence?

- Which is the canonical structure of a OOP program?
  - Hierarchical Structures
  - They are modelled using Gang of Four Composite Pattern
  - The Composite models Part/Whole Hierarchy
  - The Composite/Visitor duo

Patterns are Golden Hammers – at least for some

A Travel and Its aftermath



सत् चित् आनान्द्

Sat - Chit - Anand

Truth - Consciousness - Bliss

Existence-Essence-Bliss
Ontology-Epistemology
- Axiology

Why this session?

SAT	CHIT	ANANDA
Existence	Essence	Bliss
Ontology	Epistemology	Axiology
Who am I?	What Can I know?	What should I do?
Matter	Mind	Bliss
Structure	Behavior	Function

Three Pillars of any System

- Structure,Behvior and Function are three pillars of the system
- How these things appear in Economics?
- How these things appear in Cognitive Psychology?
- How these things appear in Computer Programming?

The Composite/Visitor Pair from OOP

- The Hierarchies are modelled using Composites (Structural)
- Processing Composites through Visitor (Behavior)
- Evaluation of Expression using AST (Structural) and Visitor
- Directory Traversal and Processing using Composite/Visitor
- Problems with the Composite/Visitor

The Problem with Composite/Visitor and its Rectification

- The Visitor needs to understand the structure of the Composites
- Is there any Pattern which helps us to navigate the content in a Structure agnostic manner?
- Flattening the Composite and navigation of the data using Iterator pattern
- Demerits of Iterator Pattern

Reversing the Gaze through Observable/Observer

- The Fire/Forget solution to the Blocking problem
- Notification to people who have subscribed
- Threads, Pools and Asyncrhony management
- Scheduling Execution

In Short, the Reactive Programming model is

- Modelling Hierarchical Structures using the Composite
- Behavioral processing using the Visitor Pattern
- Flattening the Hierarchy and navigating them through Iterator
- Notification to the subscribers using Observable/Observer pattern
- Functional Composition and Immutability
- Schedulers and Asynchrony

Case Study #2

- A Library for Evaluating Expressions (C#)
- A Stack based Expression Evaluator
- An AST based Expression Evaluator
- An Expression Evaluator based on Composite/Visitor
- An Expression Evaluator based on Iterator
- An Expression Evaluator which uses Rx

Expression Evaluator – Part #1

```
class Stack{
        double[] stk;
        int top_stack = 0;
        public Stack(){stk = new double[256];top_stack = 0;}
        public void Clear(){top_stack = 0;}
        public void push(double dbl ){
                if ( top_stack == 255 ){Console.WriteLine("Stack OverFlow"); throw new Exception();}
                        stk[top_stack++] = dbl;
        }
        public double  pop(){
                if ( top_stack == 0 ){Console.WriteLine("Stack UnderFlow"); throw new Exception();}
                        return stk[--top_stack];
                }
        }
        class RDParser : Lexer{
                 TOKEN Current_Token;
                Stack ValueStack = new Stack() ;
                        public RDParser(String str):base(str){}
                        public double CallExpr(){
                                ValueStack.Clear();Current_Token= GetToken(); Expr();  double nd = ValueStack.pop();
                                return nd;
                        }
                        public void Expr(){
                                TOKEN l_token;
                                Term();
                                if  ( Current_Token == TOKEN.TOK_PLUS  || Current_Token == TOKEN.TOK_SUB ) {
                                        l_token = Current_Token;Current_Token = GetToken();
                                        Expr();
                                        double x = ValueStack.pop(); double y = ValueStack.pop();
                                        ValueStack.push( (l_token == TOKEN.TOK_PLUS ) ? (x + y) : (y-x) );

                                }
                        }
```

Expression Evaluator – Part #1 (Contd…)

```
public void Term(){
        TOKEN l_token;
        Factor();
        if ( Current_Token == TOKEN.TOK_MUL  || Current_Token == TOKEN.TOK_DIV ) {
                l_token = Current_Token; Current_Token = GetToken();
                Term();
                double x = ValueStack.pop(); double y = ValueStack.pop();
                if ( x == 0 ) { Console.WriteLine("Division By Zero Error");throw new Exception();}
                ValueStack.push( (l_token == TOKEN.TOK_MUL ) ? (x * y) :  (y/x) );
        }
}

public void Factor(){
        TOKEN l_token;
        if ( Current_Token == TOKEN.TOK_DOUBLE ){
        ValueStack.push(GetNumber()); Current_Token = GetToken();
                        }
                        else if ( Current_Token == TOKEN.TOK_OPAREN ){
                                Current_Token = GetToken();
                                Expr();  // Recurse
                                    if ( Current_Token != TOKEN.TOK_CPAREN ){Console.WriteLine("Missing Closing Parenthesis\n"); throw new Exception();}
                                Current_Token = GetToken();
                        }
                        else if ( Current_Token == TOKEN.TOK_PLUS || Current_Token == TOKEN.TOK_SUB ){
                                l_token = Current_Token; Current_Token = GetToken();
                                Factor();
                                double  x = ValueStack.pop();
                                if ( l_token == TOKEN.TOK_SUB ) { x = -x;}
                                ValueStack.push(x);

                        }
                        else { Console.WriteLine("Illegal Token"); throw new Exception();}
                }

}
```

Expression Evaluator – Part #1 (Contd…)

```
class EntryPoint {
                public static void Main(string[] args) {
                        if ( args.Length == 0 || args.Length > 1 ) {
                                Console.WriteLine("Usage : Expr \"<expr>\" \n");
                                Console.WriteLine(" eg:- Expr \"2*3+4\" \n");
                                Console.WriteLine(" Expr \"-2-3\" \n");
                                 return;
                        }

                try {
                        RDParser parser = new RDParser (args[0]);
                        double nd  = parser.CallExpr();
                         Console.WriteLine("The Evaluated Value is {0} \n",nd );
                }
                catch(Exception e ) {
                        Console.WriteLine("Error Evaluating Expression\n");
                        Console.WriteLine(e);
                        return;
                }
        }
}
```

Who is more important?

Aaj mere paas building(C++) hai, property(C#) hai , bank balance (Java) hai, bungla(Python) hai,gaadi hai(PHP)
Kya hai, Kya hai tumhare paas?

....Mere Paas Maa (JavaScript) hai

Questions

- If any ?