1. Introducere – ce tema si de ce aceasta tema

A Tale of Two Worlds: FP and OOP este un case study in care se compara doua dintre cele mai populare paradigme din industrie: programarea functionala si programarea orientate-obiect.

OOP-ul a dominat mult industria IT pana la aparitia procesoarelor multi core, cand s-a realizat faptul ca aceasta paradigma nu se impaca bine cu multithreading/parallelism, existand problem ca deadlocks, race conditions, starvation, probleme cauzate de natura mutabila a datelor. Industria si-a amintit ca FP-ul are la baza imutabilitatea, iar o data cu asta, numeroase elemente specifice FP-ului au fost introduce in limbaje considerate OOP. Asta a dus la o confuzie din partea programatorilor, fiind probleme atunci cand se doreste sa fie o distinctie intre paradigme. Scopul lucrarii este sa diferentieze clar intre cele doua paradigme si sa clarifice eventualele confuzii.

* ~ 1:15

1. Prezentarea temei

Pentru ilustrarea conceptelor am ales problema clusterizarii ierarhice, o problema de invatare automata ce prezinta niste anumite caracteristice ce imi vor permite sa imi ating scopul.

In mare parte, algoritmul se ocupa cu gruparea unor puncte in clustere in functie de anumite criterii.

Acesta prezinta mai multe elemente:

* + Punct – 1D, 2D, 3D
  + Clustere – o colectie de puncte
  + Distanta dintre doua puncte – au fost definite 3 tipuri de distante diferite
  + Metoda folosita pentru a uni doua clustere – au fost definite 3 tipuri diferite

Pentru Implementare, am folosit Scala pentru FP, iar Java pentru OOP, oferind aceeasi implementare pentru ambele cazuri.

* ~ 1:00

1. Lines of Code

Desi nu este o metrica esentiala sau efectiv relevanta, mi s-a parut importanta in cazul de fata deoarece varianta functionala are cu aproape 50% mai putin cod scris decat varianta OOP, numarul de fisiere fiind totodata mult mai mic. Asta nu e decat un caz izolat, intrucat intr-adevar, un cod scris functional este de obicei mult mai scurt decat cel OOP.

* ~0:30

1. Modelarea datelor

ADT – in programare functionala se doreste a se defini relatii care sa se apropie de varianta matematica. Astfel ca datele sunt caracterizate doar prin proprietati, fara comportament atasat. Astfel ca un punct, caracterizat printr-un nume, poate fi unidimensional(doar X), bidimensional( X si Y), etc.

Inheritance/Polymorphism – in oop se prefera polimorfismul prin mostenire, mai exact implementarea unei interfete si folosirea acesteia in locul implementarilor efective in algoritm. Diferit de varianta functionala este faptul ca datele in oop au si comportament atasat. Vom vedea mai departe cum impacteaza acest aspect codul.

1. Aplicarea logicii peste date

Programarea functionala este despre functii – totul se invarte in jurul lor, ideea de input-output, pe cand OOP-ul este despre obiecte si behavior atasat lor.

Type Classes – foarte asemanator interfetelor din OOP, defineste un contract cu eventual anumite restrictii pe tipuri, oferind mai tarziu implementari pentru anumite tipuri dorite de catre programator. Implicit-ul din Scala este un subiect vast, si nu il voi acoperi, insa voi spune doar atat – compilatorul va cauta in implementarile oferite de catre programatorul pe aceea care se potriveste cu cea din clasa implicita si va oferi unei clase niste metode de care ea nu dispunea inainte.

Folosind type classes, codul va fi mult mai modular decat in OOP – programatorul nu este restrans sa ofere implementari in clasa propriu zisa – o poate face oriunde, iar prin importuri si cu ajutorul compilatorului, se va gasi implementarea potrivita pentru a avea parte la anumite metode/functii.

Metode – obiectele au comportament atasat. O problema cu aceasta abordare reprezinta management-ul claselor – acestea tind sa ajunga imense. De exemplu, implementarea pentru ArrayList din Java are 1.5k linii. Insa si acesta poate fi un lucru bun, cel putin cand vine vorba de performanta cat si unitate a codului. In variant cu type classes, pot aparea problem la import/gasire de implementari.

1. Algoritmul propriu-zis

Cand vine vorba de algoritmul propriu-zis, acesta incearca sa creeze clustere pana cand ramane unul singur.

Modul in care este scrisa problema este diferit totusi – OOP foloseste o varianta iterativa, in timp ce FP foloseste o metoda recursiva.

Ambele paradigme sunt influentate de ceea ce sta la baza lor – in cazul OOP-ului, mutabilitatea si pasarea/modificarii starii, iar in cazul FP-ului, imutabilitatea.