Intrebari PP

Curs 1

BNF stands for **Backus-Naur Form**. It is used to write a formal representation of a context-free grammar. It is also used to describe the syntax of a programming language. EBNF means Extended BNF. There's not one single EBNF, but each author or program define their own variant that's slightly different. **BNF** syntax can only represent a rule in one line, whereas in **EBNF** a terminating character, the semicolon, marks the end of a rule.

1. Programare structurata

-poate fi vazuta ca un subset al programarii procedurale

-orice program poate fi realizat prin combinarea structurilor de control: secventa, selectia

si iteratia, iar fiecare structura de control are o singura intrare si o singura iesire

2. Programare procedurala(programare imperativa)

-bazata pe conceptul de apel procedural

-proceduri numite uneori rutine/subrutine sau functii care sunt similare cu cele folosite

in programarea functionala care contin o serie de pasi de executat

-schimbul de date intre functii se realizeaza prin transfer de parametri(argumente de intrare

si valori rezultate in urma executiei procedurii)

3. Programarea nestructurata

-bloc continuu

-salt direct

4.Programare bazata pe componente

-similara oop dar cu o abordare bottom-up(asamblarea subprogramelor intr-un program mai mare

functional)

-diferenta fata de oop: in loc sa se proiecteze conform unui model descris de proiectant se

folosesc componentele disponibile pentru a asambla o solutie viabila

5.Programarea orientata obiect

-obiectele POO sunt de obicei reprezentari ale obiectelor din viata reala a.i.programele

realizate prin tehnica POO sunt mai usoare de inteles, de depanat si de extins decat programele

procedurale

6.Programare modulara

-structura fiecarui modul trebuie sa fie suficient de simpla pentru a putea fi complet

inteleasa

-detaliile sistemului care se presupune ca se vor modifica independent vor fi plasate in

module diferite

-singurele legaturi intre module vor fi acelea a caror modificare este improbabila

-orice structura de date este incapsulata intr-un modul; ea poate fi accesata direct din

interiorul modulului, dar nu poate fi accesata din afara modulului decat prin intermediul

variabilelor globale continute in acel modul

-principiul de baza pt module: incapsularea

Un modul are doua componente: interfata si implementarea

Programare bazata pe aspect(din oop)

-AOP reprezinta o noua METODOLOGIE care vine ca o COMPLETARE la OOP, aducand cateva principii

noi care imbogatesc aceasta paradigma de baza

-NU ESTE O NOUA PARADIGMA, ci o simpla completare la OOP

7.Programare bazata pe constrangeri

-relatiile intre variabile pot fi descrise sub forma de constrangeri

-sunt asociate cu programarea logica, functionala si imperativa

8. Programare orientata pe concept

-un concept este alcatuit dintr-o clasa de obiecte si o clasa de referinte

-va folosi relatia de incluziune a conceptelor in loc de mostenire

-principalul element constructiv este conceptul (metode de afaceri si reprezentari si accesul

acestora)

9.Programare bazata pe eveniment

-in loc sa se astepte completarea executiei unei comenzi pentru a procesa informatia, in

acest caz sistemul este preprogramat cu o bucla pe eveniment

10. Programare declarativa

Are doua intelesuri:

1. un program este declarativ daca descrie "ceva" si se pune mai putin accent despre cum este

creat acel ceva

2. un program este declarativ daca este scris intr-un limbaj care suporta modelul programarii

functionale, logice sau cu constrangeri

11.Programarea imperativa

-este un termen in opozitie cu programarea declarativa care descrie realizarea operatiilor de

calcul folosind termenii de stare si declaratie/instructiune care va conduce la modificarea

starii

12. Programare functionala

-realizeaza sinteza automata a tipurilor ->depistarea erorilor subtile generate de declararea

si utilizarea eronata a tipurilor datelor prelucrate

-trateaza orice calcul ca o evaluare a functiilor matematice

-limbajele functionale au un suport matematic solid si un grad ridicat de abstractizare

Intrebari curs 2

Constant folding is the process of recognizing and evaluating constant expressions at compile time rather than computing them at runtime. Terms in constant expressions are typically simple literals, such as the integer literal 2, but they may also be variables whose values are known at compile time

M-n theorem: In practical terms, the theorem says that for a given programming language and positive integers *m* and *n*, there exists a particular algorithm that accepts as input the source code of a program with *m* + *n* free variables, together with *m* values. This algorithm generates source code that effectively substitutes the values for the first *m* free variables, leaving the rest of the variables free.

1.Model church-turing

R: Spune ca un calcul poate fi facut de o masina doar daca este calculabil in sens Turing. Orice problema de calcul este calculabila daca se incadreaza in una din categoriile: general recursive functions, lambda-computable sau Turing computable(de fapt toate 3 sunt echivalente)

2.Care sunt problemele masinii virtuale java sau c# ?

R: Performanta (datorita interpretoarelor) este 1-3 ori mai lenta decat implementarile similare in C(unsafe). Java e pe cale de disparitie, dar poate exista ptc sunt deja multe aplicatii in java(20 ani)

3.Ce este polyglot?

R: Polyglot este un limbaj de programare care are mai multe interpretoare(graal etc) si practic poti folosi pentru a rula mai multe limbaje. Se foloseste de AST pentru a scapa de diferentele de sintaxa

4.Implementare AST? De ce e mai buna decat o masina virtuala?

R: ast = abstract syntax tree; utilizare ast = ascunde diferentele de tratare a fiecarui limbaj; ajuta sa punem cap la cap instructiunile din toate limbajele -> nu mai conteaza din ce limbaj vin instructiunile

5.Ce este hotspot? Ce este graal?

R: Graal este interpretorul lui Polyglot. Hotspot este o masina virtuala java. Graal este scris in Java si se bazeaza pe Hotspot care e scris in C

6.Care sunt avantajele Graal fata de hotspot?

R: hotspot este doar proiectul initial, graal e bazat pe hotspot si aduce alte functionalitati: libraria truffle(care permite accesul la limbaje precum R,Ruby,js etc) + tool-uri + compiler nou ===> avantaj de timp la rularea programelor

7.Care sunt limbajele suportate de Graal la ora actuala?

R: js,ruby,r,python,java

8.Ce este calculul functional?

R: **se bazeaza pe** evaluarea de funcții matematice și evită starea și datele mutabile. Se pune accent pe aplicarea de funcții.

9.Ce este o functie lambda?

R: Putem crea o functie fara sa le asignem un nume explicit(functii fara nume)

10.Explicati functiile Curry

R: functii cu argument multiplu pot returna mai multe valori + functii ce primesc serial argumentele(primul apel are n parametri, urmatorul n-1, urmatorul n-2 pana se ajunge la functia cu un parametru care returneaza o valoare = recursivitate)

A curried function is a function that takes multiple arguments *one at a time.* Given a function with 3 parameters, the curried version will take one argument and return a function that takes the next argument, which returns a function that takes the third argument. The last function returns the result of applying the function to all of its arguments.

11.Diferenta dintre tipurile dinamice si tipurile statice de date?

R: tipuri statice = fiecare variabila este bine legat de un tip de data(de ex int c++). NU SE POATE SCHIMBA TIPUL UNEI VARIABILE IN TIMPUL PROGRAMULUI; tipuri dinamice = de ex python

12.Limbaje ce suporta tipuri "tari" de date si tipuri "slabe" de date

R: python suporta ambele(de ex poti face int(1) + float(2.3), dar nu poti 1 + "2.3"), javascript(tip slab de date) de ex: 1 + "2" = "12" -> face conversii implicite

13.Enumerati tehnicile curente de optimizare a unui cod

R: reorganizare noduri AST, evaluare partiala

14.Reorganizarea AST? Avantajele utilizarii acestei metode

R: Reorganizarea unui AST practic schimba structura arborelui astfel incat sa se evite pasi in plus, sporind practic eficienta codului

15.Ce inseamna evaluarea partiala?

R: Evaluezi diferite parti ale programului in valoarea optimizarii(de exemplu partile pe care deja le stii inainte de compile time, codul)

In computing, **partial evaluation** is a technique for several different types of program optimization by specialization. The most straightforward application is to produce new programs that run faster than the originals while being guaranteed to behave in the same way.

16.Ce este un evaluator partial?

R: Un evaluator partial particularizeaza programele. De exemplu, daca un program este o functie prog definita pe date intrare(cunoscute la compile time) + date ce urmeaza a fi introduse in program de utilizator cu valori in Output, atunci un evaluator partial transforma programul intr-o versiune noua ce incapareaza datele deja cunoscute, specializand practic programul si facandu-l mai eficient(eventual se rescriu liniile de cod)

17.La ce se refera specializarea unui program?

R: Pentru un set de date(in1) programul se specializeaza, scriindu-se mai eficient instructiunile(de ex trecere de la program structurat la program nestructurat)

18.Optimizare in Graal?

R: Mai putine instructiuni datorita evaluarii partiale

19.Metode depanare Graal?

R: Loggings(trimitere de mesaje), mecanismul try-throw-catch

20.Ce sunt metricele unui program si de ce ne-ar trebui?

R: De ex: memorie RAM utilizata, viteza de executie, memorie utilizata(octeti),numar linii cod !!CounterKey(timp scurs). Poate gasesti leaks de memorie

21.La ce se refera dumping-ul?(poate unui executabil)

R: Se ingheata memoria pt functia care s-a oprit incorect(a crapat) din functionare si se salveaza datele cu scopul de a face debug pe ele

The key **difference between** the two is that with **static type checking**, the **type** of variable is known at compile time (it **checks** the **type** of variable before running) while with **dynamic type checking**, the **type** of variable is known at runtime (it **checks** the **type** of variable while executing).

lambda calculabila = calculabila conforma modelului turing

mutable types = cand dorim sa ii asociem o noua valoare pur si simplu se schimba(de exemplu lista in python)

immutable types = cand dorim sa ii asociem o noua valoare, de fapt se creeaza un nou obiect(cu alt id) spre care variabila noastra pointeaza

dynamic typing = nu e nevoie sa declari tipul obiectului, isi da el singur seama(ex python: a = 7, isi da seama ca e vorba de un int); fac type-checking la run-time

static typing = trebuie sa declari tipul obiectului la declarare(ex: int a = 7); fac type-checking la compile time

strongly typed = nu face conversii implicite(de exemplu 1 + "2" = eroare)

weakly typed = face conversii implicite(de exemplu in javascript: 1 + "2" = "12")

IntrebariCurs3

1.Ce este descompunere functionala dpdv al oop?

Inseamna ca atunci cand proiectam un program sa ne facem un plan, sa distingem niste functionalitati iar apoi sa le modulam si sa le algoritmizam impartindu-i intr-un numar de pasi

Ex: Sa se acceseze descrierea unor forme existente intr-o baza de date apoi sa se afiseze aceste forme

1. Identifica lista de forme in baza de date

2. Deschide lista de forme din baza de date

3. Ordoneaza lista de forme conform cu un set de reguli

4. Afiseaza formelele pe monitor

2.Care sunt principalele probleme ale specificatiilor de la client?

R: Clientul habar n-are ce vrea si da info incomplete, gresite, ii mai vin idei pe parcurs, greseli de comunicare/intelegere

3.Care sunt motivele pt care clientul nu da ce trebuie?

R: Diferente de comunicare, clientul omite informatii care crede el ca sunt evidente desi sunt relevante pentru proiectant, nu se gandeste inainte la toate aplicatiile

4.Ce efect au coeziunea scazuta si cuplarea stransa?

coeziune - cat de apropiate sunt operatiile dintr-o metoda

cuplarea - cat de stransa este legatura dintre doua metode

integritate interna(coeziune stransa)

relatii directie, vizibile, flexibile si directe(strans cuplate)

Coeziunea scazuta = operatiile dintr-o metoda sunt strans legate intre ele si o modificare mica se propaga peste tot

5.Care sunt pasii din proiectarea OOP?

R: Izoleaza obiectele din lumea reala, abstractizeaza obiectele, determina responsabilitatea grupului fata de alte grupuri

6.Ce este brainstorming-ul?

R: aruncari spontane de idei, persoanele care participa ar trebui sa stie pt a da idei pertinente

7.Ce pasi implica metodologia de proiectare OOP?

brainstorming - pentru a localiza clasele posibile, se considera toate ideile

filtrarea claselor - pt a gasi duplicatele si a elimina pe cele in plus

scenarii - necesare pentru a fi siguri ca s-a inteles colaborarea intre obiecte

algoritmi - sunt proiectati pentru a defini toate actiunile pe care clesele trebuie sa le poata efectua

8.La ce se folosesc fisierele CRC?

R: Pt a mentine informatiile primare despre clase,subclase, superclase, responsabilitati, colaborari, cum relationeaza intre ele clasele

9.Care este flow-ul general pt rez si proiect unei probleme in OOP?

abstractizare, proiectare, implementare

?

R: Se grupeaza elementele in clase

10.Cum creeaza Kotlin obiectele asociate unei clase?

R: Instantiere prin constructori

11.Care este rolul lui this?

R: Ne ajuta sa ne referim la obiectul respectiv(in metodele unei clase de ex)

this -> instanta curenta

cand vrem sa ne referim la instanta externa

12.Care este rolul lui scope?

R:evita conflictele de nume

13.Cum se poate face controlul fluxului de executie ca o expresie?

R: cu if,else, try,catch, when

14.Ce este NULL? Ce rol are NULL?

? R: Null este un pointer la nimic. Null nu se poate converti la un tip

15.Explicati verificarea si conversia de tip in Kotlin

R: Implicita(dar poate fi facuta explicita).

Verificam cu "is" = "instance of"(Java)

Kotlin face cast a unei variabile la ultima verificare de tip

16.Cum se poate utiliza when ca un switch case?

R: In loc de default se pune else; la case se pune doar valoarea pe care sa o ia variabila din when()

17.Cum se poate utiliza when ca o expresie?

? R: Asignam valoare unei variabile cu when

18.Ce este clasa anonima?

R: Este o clasa ce se poate utiliza pt apeluri singulare si nu este recomandata

Object expressions create objects of anonymous classes, that is, classes that aren't explicitly declared with the class declaration. Such classes are handy for one-time use. You can define them from scratch, inherit from existing classes, or implement interfaces. Instances of anonymous classes are also called *anonymous objects* because they are defined by an expression, not a name.

19.Ce sunt clasele pt gestiunea datelor?

R: Clasele de tip enum si data(pastreaza date)

20.Ce sunt metodele statice in Kotlin?

R: Nu avem in clase metode statice(in Kotlin). La nivel de pachet avem metode statice.

21.Ce sunt obiectele companion si unde se folosesc?

R: Functiile companion care apartin unei clase iau locul metodelor statice, putand fi apelata fara un obiect instatiat

Companion objects are same as “object” but declared within a class.

The main difference between an object and companion object is, In object, methods and properties are tied to class’s instance but in the case of companion object, they are tied with the class.

22.Suporta Kotlin mostenirea simpla direct?

R: Mostenirea multipla se simuleaza

Se suporta mostenirea simpla directa. Clasele fara parinte vine de la clasa de baza(object, aia de baza)

Pt a face o clasa derivabila avem cuvantul cheie "open"

The **open** keyword means “**open** for extension“: The **open** annotation on a **class** is the opposite of Java's final : it allows others to inherit from this **class**. By default, all **classes** in **Kotlin** are final.

23.Ce sunt functiile cu expresie unica?

R: Functie ce contine o simpla expresie si folosind inferenta de tip isi da seama pe baza evaluarii expresiei ce tip are de returnat

24.Ce sunt functiile membru?

R: Functii in interiorul unei clase, obiect sau interfata.

25.Cand sunt recomandate functiile locale?

R: Functii mici, definite in interiorul altor functii

Cand iesim din scope-ul in care au fost definite nu le mai putem folosi

Dorim sa ascundem detalii de implementare a unei functii mai mari

Cand nu mai avem nevoie de ele in afara scope-ului

26.Ce sunt functiile top-level?

R: Functiile globale din C++

Toplevel functions are **functions** inside a **Kotlin** package that are defined outside of any class, object, or interface. This means that they are **functions** you call directly, without the need to create any object or call any class.

27.Ce este list cromprehension si cum se poate face in Kotlin?

R: Facem liste pe o linie, e mai rapid, am facut in graba

izoleaza obiecte din lumea reala, abstractizeaza obiectele care au prop si comportamente similare

determina resp grupului dpdv al interactiunii cu alte grupuri

Metodologie POO

brainstorming - pentru a localiza clasele posibile

filtrarea claselor - pt a gasi duplicatele si a elimina pe cele in plus

scenarii - necesare pentru a fi siguri ca s-a inteles colaborarea intre obiecte

algoritmi - sunt proiectati pentru a defini toate actiunile pe care clesele trebuie sa le poata efectua

In named parameters, we name the passing parameters with labels and then even if we change the parameters passing order in the function. The values are assigned in the correct order. In short, the sequence does not matter.

-flow ul general pt implementarea unei probleme in oop

abstractizare, proiectare, implementare

-probleme aduse de utilizarea null ului

verificare, null safety

-cum rezolva kotlin problema legata de null

operator de siguranta ?

-explicati verificarea si conversia de tip in kotlin

In Kotlin

```
var value1 = 10
val value2: Long = value1  //Compile error, type mismatch
```

However in Kotlin, conversion is done by explicit in which smaller data type is converted into larger data type and vice-versa. This is done by using helper function.

```
var value1 = 10
val value2: Long = value1.toLong()
```

-ce este clasa anonima

clasa fara nume, object={}, nu se recomanda

-permit clasele acestea o gestiune avansata?daca da, cum?

(functii cu procesari in aceste clase, este posibil?)

-ce sunt metodele statice in kotlin

functii ce pot fi apelate fara a instantia un obiect, apel folosind clasa

functiile statice globale sunt locale in interiorul acelui fisier unde sunt declarate

-kotlin suporta mostenirea simpla direct?

da, si multipla

-ce sunt functiile cu expresie unica

= intre functie si definitie, asemanator functiilor inline

-cand sunt recomandate functiile locale

Local functions can also access local variables of outer functions and help with code reuse.

-ce este list comprehension si cum se poate face in kotlin

A list comprehension is a special syntax in some programming languages to describe lists, functii ex listof, etc

IntrebariCurs4

1. Cum definiti proiectarea aplicatiilor software?

Proiectarea software reprezinta un proces de rezolvare a unor probleme, obiectivul fiind sa se gasesaca si sa se descrie o cale de a implementa necesitatile functionale ale sistemului, tinand cand de constrangerile cleintului

2. Care e diferenta intre must have si nice to have ?

produsele principale(must have), cele secundare(nice to have)

3. Definiti o componenta?

Entitate fie fie soft fie hard care are un rol bine determinat si poate fi inlocuita cu alta componenta

4. Din ce este alcatuit un sistem daca il analizam din prisma analizei modulare?

Subsisteme, componente si module

5. Ce este modelul domeniului?

System,susbsysteme implementat de Component, care este definit la nivelul limbajului de catre module si framework

6. Ce este UML(unified modeling language)

Este un limbaj grafic adoptat mondial folosit pentru reprezentarea, vizualizarea si documenatarea componentulor unui sistem software de dimensiuni mari

7. Ce este modelul de proiectare si implementare in cascada? AVANTAJ

Fiecare nou proces se implementeaza dupa ce acela anterior este complet realizat si testat.

Etape: specificarea cerintelor, analiza sistemului, proiectarea sistemului, implem sistemului, testarea sistemului, instalarea sistemului,

mentenanta sistemului.

8. In ce conditii se pot suprapune AGILE si WATERFALL?

In waterfall, o etapa viitoare nu poate incepe inainte ca o etapa din trecut sa se fi incheiat (o etapa noua poate depinde de o etapa din trecut)

In AGILE, noile etape depind de cele vechi, dar se proiecteaza simultan (pot depinde unul de celalalt in paralel)

9. Enumerati modelele sistem :

Modelul obiect (str sistemului, obiectele si relatiile)

Modelul functional (fluxuri de date din sistem)

Modelul dinamic (cum reactioneaza sist la stimuli externi)

10.Enumerati modelele task-urilor: __,planificarea

Harta PERT(care sunt legaturile dintre task-uri),

Planificarea(cum poate fi aceasta realizata in durata de timp rezervata?)

Harta organizationala(care sunt rolurile in proiect?)

11.Eumerati mecanismele centrale ale unei aplicatii OOP

Encapsulation, Abstraction, Inheritance, and Polymorphism.??

12. Cum se defineste un model dpdv al UML:

Un model este o descriere completa a unui sistem dintr o perspectiva particulara

12') Tipuri de proiectare specifice

R: proiectare arhitecturala

proiectarea claselor

proiectarea interfetei vizuale

proiectarea bazelor de date

proiectarea algoritmilor

proiectarea protocoalelor

13. La ce este buna abordarea bazata pe componenete in proiectare?

R: Componentele pot fi inlocuite ulterior cu unele mai bune,

acest lucru fiind posibil datorita nivelului ridicat de incapsulare.

14. Dati exemple de aplicare a principiului cresterii coeziunii

Grupeaza ce este apropiat cat mai mult(doua clase care comunica trebuie incluse in acelasi pachet)

15. Exemple de aplicare a principiului reducerii cuplarii(cuplare = interdependente in

De exemplu: reducem numarul de dependente intre clase, daca fiecare

clasa ar fi un nod intr-un graf ne-am dori sa minimizam numarul de arce,

sa fie cat mai independente pentru a ne fi usor sa modificam lucrurile

ulterior

16. La ce se refera principiul de baza in gestiunea abstractiilor?

R: ascund padurea ca sa nu ma incurc de copaci

entitate generica rezultate din analize (nu e necesar sa stim detalii

de implementare (tip de date))

17. Care sunt principiile complementare reutilizarii?

proiecteaza pentru reutilizare, proiecteaza prin reutilizare

18. La ce se refera proiectarea pt flexibilitate?

Anticiparea activa a modificarilor pe care un proiect le poate suferi

si pregatirea din timp pt acestea

R: Anticiparea schimbarilor ce pot aparea pe parcurs. (reducerea

cuplarii + cresterea coeziunii

19. Cum se gestioneaza problemele de inlocuire sau disparitie a unor componente

puse la dispozitie de o tehnologie sau framework?

R: De ex, in cazul inlocuirii, se asigura backwards compatibility prin

respectarea principiilor SOLID (Liskov substitution)

20. La ce se refera proiectarea pt portabilitate? Programam aplicatia astfel incat sa fie cat mai portabila pe mai multe sisteme de operare

Utilizarea de limbaje interpretate(de ex Java), ce nu sunt dependente

de arhitectura hardware

21. La ce se refera proiectarea pt testabilitate?

Iei masuri pentru usurarea testarii

Masuri precum:

- utilizarea design patternurilor

- utilizarea unor feature-uri care acceota linie de comanda

21') Scopurile proiectarii OOP

-> usurinta in modificare sau adaugarea unor noi functionalitati

-> gestionarea independentei intre clase si pachete

-> scaderea motivelor pt care modelul s-ar putea schimba daca ar aparea

schimbari in clase/pachete

22. La ce se refera coeziunea claselor?

plasarea metodelor care opereaza pe anumitr date cat mai aproape de

datele respectivr

23. Ce este principiul de raspundere unica?

Fiecare clasa ar trebui sa faca o singura chestie

24. Ce este principiul separarii interfetelor?

Clientii nu trebuie sa fie obligati sa depinda de interfete de care

nu au nevoie

25. Principiul inchis/deschis?

Clasele tre sa fie deschise la extindere si inchise la modificare

26. Ce este BPP-ul?

R: Bune practici de proiectare (Divide and conquer)

27. Principiul substitutiei Liskov?

Clasele derivabile trebuie sa fie substituibile in clasele de baza.

Adica clasele derivate trebuie sa aduca functionalotati noi, nu sa le modifice pe cele vechi

28. Principiul dependentei inverse?

Modulele de nivel arhitectural superior nu trebuie sa depinde de cele de nivel inferior. Ambele trebuie sa depinda de abstractii, care la randul lor nu depind de implementarile concrete

29. TDD = Test-Driven Development

scrie(un test)

executa(toate testele)

scrie(codul tinta)

executa(testele)

refactorizeaza(codul)

IntrebariCurs5

1. Ce este un eveniment?

Un semnal generat de catre o aplicatie, un sistem hardware.

Poate fi definit ca un tip de semnal generat de cate program.

Indica ca s-a intamplat ceva

Ex: buton de mouse, miscare de mouse,

2. Prezentati o posibila maniera de gestionare a unui eveniment, inclusiv OS-ul

Tastatura,Mouse -> OS -> coada evenimente -> programe -> OS -> monitor

3. Cum se gestioneaza dpdv arhitectural un eveniment?

Se baga intr-o coada de evenimente si cand trebuie sa fie tratete

se scot pe rand din coada

??

4. Care este diferenta dintre evenimentele sincrone si cele asincrone?

Sincron : trimit mesaj, astept confirmare

asincron : trimit toate mesajele, nu mai astept confirmare

mouse tastatura : asincron???

5. Care sunt sursele comune pt evenimentele sincrone si asincrone?

Mouse-ul si tastatura

6. Cum se trateaza un eveniment dpvd al programatorului?

Se trimite la distribuitor care le trimite la un anumit gestionant

Polling (citire secventiala intr-o bucla infinita a tuturor

comenzilor dispozitivelor de intrare)

7. Care sunt avantajele procesarii orientate pe evenimente?

mai portabile

permit tratarea rapida a erorilor

se po folosi in time-slicing

incurajeaza reutilizarea codului

merge mana in mana cu oop

8. La ce se refera EDP si care sunt componentele principale implicate?

Event driven programming

Generatoare de evenimente

Sursa evenimentelor

Bucla de evenimente

Gestionari de evenimente

Event mapper

Inregistrarea evenimentelor

9. Ce cuprinde diagrama de secventa specifica EDP?

AppUser - actiuni

Event Object - genereaza evenimente

Event Mapper - inregistreaza gestionarul de evenimente si distribuie

Event Handler - proceseaza evenimentele

utilizatorul face diverse actiuni, are nevoie de un obiect,

obiectul genereaza evenimente, acestea ajung la mapper care

inregistreaza gestionarul de evenimente si apoi ajung in handler

10. Ce este dispecerul in EDP?

Cel care controleaza evenimentele

Distribuie evenimentele

primeste fiecare eveniment, il analizeaza pentru a-i determina tipul

si le trimite catre fiecare gestionar asociat tipului respectiv,

11. Ce inseamna wysiwyg si care este legatura cu EDP?

What you see is what you get, toata interfata vizibila

utilizatorul interactioneaza si produce evenimente

12. Explicati look-n-feel?

stabileste interactiunea vizuala a evenimentelor

look-maniera in care e facuta interfata grafica

feel-maniera in care interfata raspunde la evenimentul

dat de utilizator

13. Care sunt criteriile minimale care trebuie implementate de un GUI pentru a se mula pe utilizator?

ghideaza utlizatorul, are o anumita eleganta, sa ofere informatii ajutatoare

si sa permita utilizatorului sa faca greseli

14. La ce se refera pick correlation?

procesul de selectie a unei ferestre sau aplicatii care trebuie sa trateze un efect oarecare

15. Ce este un GUI chat?

16. Explicati MVC-ul

Model view controller - modelezi datele, view inseamna cum arati datele, controller da operatii modelului

Controler modifica starea modelului, cand se schimba starea view-ul afiseaza noua stare

17. Ce este SDL si unde se foloseste?

Este o biblioteca folosita la creare de software

SimpleDirectMediaPlayer

pt dezvoltarea jocurilor

18. Care sunt modulele SDL si echivalentele lui cu DIVX?

Video - DriectDraw

Gestiunea evenimentelor - DirectInput

Joystick - DirectInput

Audio - DirectSound

CD-ROM - NU are echivalent

Firul de executie - NU are echivalent

Timere - NU are echivalent

19. Ce este list comprehension? Cele mai comune utilizari

constructie ce se bazeaza pe creearea unei liste folosind un iterabil(generator, lista)

Scriem liste/structuri multidimensionale pe o linie

list1 = [i**2 for in range(0,8)] (pentru for)

list1 = [i for i in range(0,8) if(i%2==)] (pentru if else)

list1 = [i**2 if (i%2==) else i**3 for i in range(0,8)] (if else)

//matrice : list1 = [j for j in range(0,6) for i in range(0,3)]

20. Ce este un eveniment virtual? Cum se trateaza in tkinter?

You can create your own new kinds of events called *virtual events*. You can give them any name you want so long as it is enclosed in double pairs of **<<…>>**.

21. Ce este si unde este necesara organizarea matriceala a entitatilor intr-un GUI python?

Fiecare element este pe o linie si o coloana.

22. Explicati maniera arborescenta de creare a meniurilor in tkinter

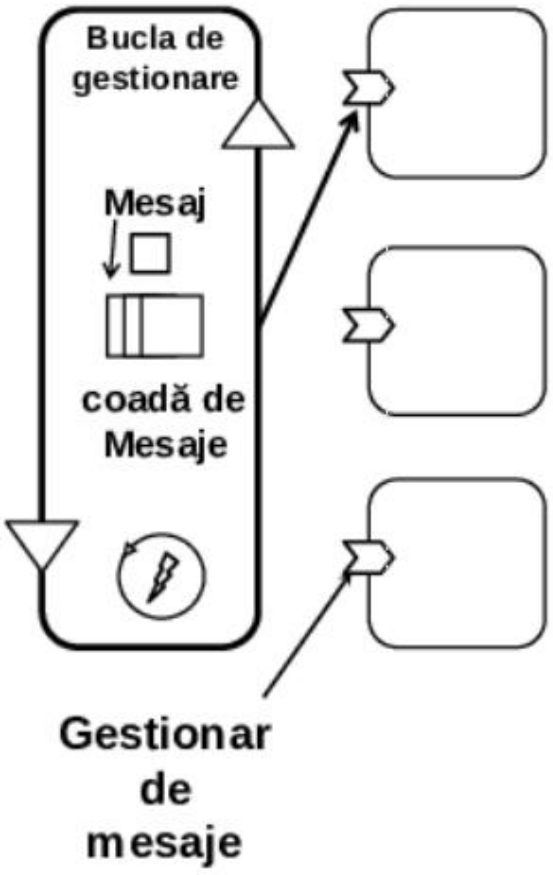
Creez submeniurile si apoi le asamblez intr-un meniu mare

23. Ce ar trebui urmarit cand trebuie tratat un eveniment Frame

???

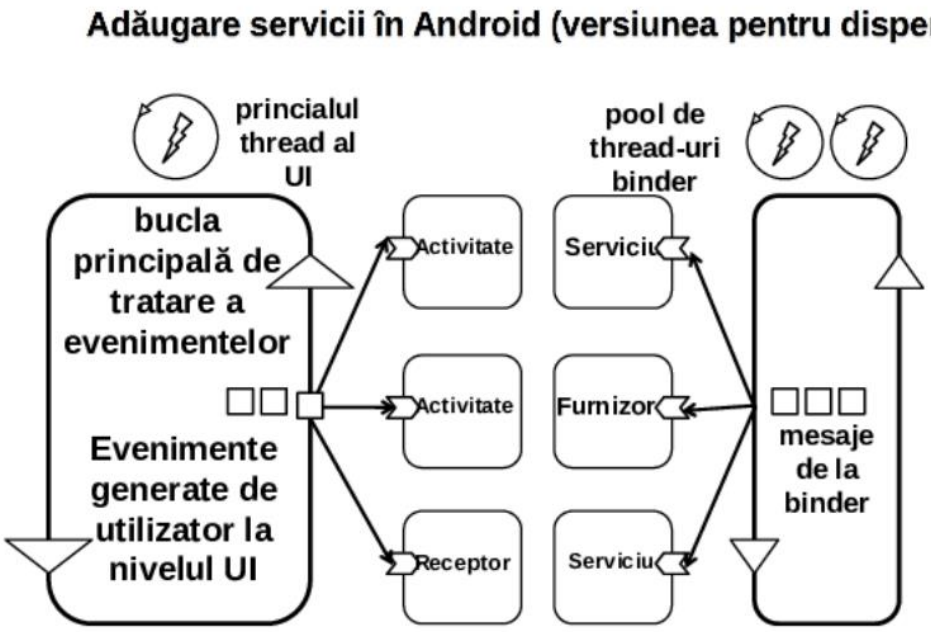
24. Explicati tratarea evenimentelor in Android

The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.



25. Cum se pot adauga servicii in Android?

To create a background service, first you need to add the service into your manifest file. Then, create a class that extends service. Finally, in your activity start the service.



26. Ce est un widget?

obiect din cadrul unei interfecte grafice orientata obiect(fereastra text)

27. Ce tip de aplicatie are specific mvc ul?

aplicatii web

28. Ce este mvp?

se refera la minimum valuable product-status al aplicatiei care poate fi folosita si testata pentru a o imbunatati dupa un feedback

29. Pe langa list comprehension ce mai exista din categoria asta?

Dictionary Comprehensions:
Extending the idea of list comprehensions, we can also create a dictionary using dictionary comprehensions. The basic structure of a dictionary comprehension looks like below.

output_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}

Set Comprehensions:
Set comprehensions are pretty similar to list comprehensions. The only difference between them is that set comprehensions use curly brackets { }. Let's look at the following example to understand set comprehensions.

Generator Comprehensions are very similar to list comprehensions. One difference between them is that generator comprehensions use circular brackets whereas list comprehensions use square brackets. The major difference between them is that generators don't allocate memory for the whole list. Instead, they generate each value one by one which is why they are memory efficient. Let's look at the following example to understand generator comprehension:

30. Ce este modelul binder?

The binding function is used to deal with the events. We can bind **Python's Functions** and methods to an event as well as we can bind these functions to any particular widget.

Polling : bucla infinita

Interrupt-driven : asteapta aparitia unor intreruperi

Sincron : trimit mesaj, astept confirmarea ca primul a ajuns

apoi trimit al doilea mesaj. eu rezerv resurse care sa

primeasca evenimentele

asincron : trimit toate deodata nu astept confirmare

nu ne mai batem capul daca va fi receptionat si tratat

nu se mai blocheaza aplicatia sa astepte confirmari

Intrebari Curs6:

1.Enumerati domeniile posibile de utilizare python

Backend, front end, inteligenta artificiala, securitate, aplicatii web

hardware low level, criptografie(nu chestii serioase)

2. Care sunt paradigmlele de programare ale lui Python?

functionala

orientate obiect

imperativa

procedurala

3. Ce tipuri de conversii explicite pt tipurile de date suporta python?

integer,string,complex, floating point, long

4. Explicati instructiunea try-except

except exception

5. Ce este o exceptie?

Este un eveniment care apare in timpul executiei programului

si care opreste fluxul normal al instructiunilor. Cand programul

gaseste o situatie care nu-i convine trimite o exceptie.

Exceptia e un obiect python care reprezinta o eroare

6. La ce este utilizata pass?

De exemplu pentru a spune unei functii ca va fi implementata mai tarziu

7. Ce diferenta sunt in python fata de C in termeni de operatori simpli?

putere, //, membership si identitate la op pe biti

in python nu avem incrementari

8. Ce tipuri de operatori de atribuire sunt suportati in python?

=,+=,*=,%=,/= etc

9. Diferenta dintre op de apartenenta si cei de identitate

daca un obiect e intr-o secventa

daca au aceeasi referinta

10. Ce standard de caractere este implicit in python?

unicode

11. De ce se poate executa un cod in interiorul unui sir in python?

Deoarece python este interpretat, interpretorul merge linie cu linie

cu exec si eval

12. Ce este eval?

Evalueaza un string ca o expresie python si returneaza rezultatul

13. Care e diferenta dintre o tupla si un dictionar?

Dictionarul e mutable, tupla e immutable

14. Care sunt atributele unui obiect de tip fisier?

closed(),mode(),name(),softspace(),next(),read(),

readline(),seek(),tell(), write()

15. Care e metoda corecta de tratete a op cu fisiere?

Deschidem fisierul intr-un try tratem erorile si

inchidem la final fisierul

Daca sunt exceptii le tratam cu except

16. Cum se trateaza apelul unei functii in python?

Se opreste executia, se trimit valorile la parametrii

formali, functia isi face treaba si returneaza valoare

17. De ce pot intoarce valori multiple in python?

Cu tuple, poate sa despacheteze tuple(iterable unpacking)

18. Cum pot simula transferul prin referinta la iesirea din functie in python?

tablouri, lista

19. La ce ne folosesc cuvintele cheie utilizate ca parametru in python?

ca sa fim mai expliciti

20. Care sunt diferentele dintre C++ si suportul primar de OOP oferit de python?

this vs self(se trimite implicit, dar trebuia primit explicit)

constructorul este __init__()

indentare la python

in python avem conceptul ala Duck typing si astfel exista cumva o interfata

generata automat

21. Cum se poate adauga rapid persistenta in python?

fisier, baza de date, shelve

ce e persistenta?

stocam ceva foarte rapid care sa fie valabil in doua sesiuni

22. De ce as fi nevoit sa construiesc un GUI de la 0 in python?

vrei sa modifici comportamentul default al acelui GUI

Ex: Schimbare buton la hover in loc de click

(de exemplu vrei sa schimbi background-ul si nu iti permite GUI existent)

Intrebari Curs 7:

1. Ce sunt functiile parametrizate?

Functii bazate pe generice(argumentele sunt parametrizabile)

Any(Kotlin) = Object(Java)

Unit(Kotlin) = void(Java)

2. Ce sunt tipurile parametrizate?

Ca si containerele din C++, de ex List<Int>

parametrii au un tip(se specifica intre paranteze ascutite)

3. La ce se refera polimorfismul limitat superior?

Nu putem duce orice tip de date in orice tip de date

De ex pt comparable: se limiteaza la subgrupul asteptat

care contine elemente ce pot fi comparate

R: Se refera la limitarea tipului de date la subtipurile celui mentionat.

R: In Kotlin restrictioneaza tipurile de parametri la subclasele unei anumite clase.

4. Ce sunt limitarile superioare multiple?

sunt ca cele superioare doar ca trebuie introdusa clauza where

Limitare multiple: Ex: fun<T> minSerializable(first: T, second: T):T

where T: Comparable<T>, T: Serializable

-> se foloseste clauza "where"

5. Ce sunt tipurile generice de date?

Tipuri de date paramatrizabile(au un parametru <T> care

poate fi inlocuit cu Int, Double sau chiar ADT-uri)

ex Class<Int>

6. Modificatorii de tip?

out -> poate fi folosit doar ca tip de return

in -> poate fi folosit doar ca parametrii la functii

7. Ce este declaration site variance?

varianta specificata in momentul in care declaram o functie, restrictionez ce pot face cu parametrii

abilitatea de a specifica variance annotation la declararea clasei

iesire -> produs(out) -> covarianta

intrare -> consumat(in) -> contravarianta

7'. Ce este variance annotation?

un modificator aplicat unui parametru/argument generic, cu scopul de a

i declara varianta. In Kotlin sunt 2 variance annotations: out si in

8. Explicati problema claselor duale producator - consumator

R: Clase care folosesc un parametru covariant out si un parametru

contravariant in.

9. Explicati type projection

E util atunci cand cineva a declarat o clasa invarianta si eu

am nevoie sa fie folosita intr-un mod covariant sau contravariant.

Practic specificam tipul de varianta la utilizare

* in loc de in: nu intra nimic

* in loc de out: returneaza orice sau nimic (any?)

Type projection is **kind** of like that - we take an existing object and reduce it to just the attributes or operations that we need at that place in the code. By reducing the object to a new form, you can allow subtyping that would not otherwise be possible.

11. Ce sunt functiile generice?

Sunt functii ce au si ele la randul lor tipuri parametrice

12. Ce sunt constrangerile generice?

Limitarea superioara a constrangerilor

13. Ce este type erasure? (pierderea tipului)

la rulare instantele de tip generic nu detin nicio informatie cu privire la tipul lor, s-a sters informatia de tip

Type Erasure. As with Java, **Kotlin's** generics are **erased** at runtime. That is, an instance of a generic **class** doesn't preserve its **type** parameters at runtime. For example, if we create a Set<String> and put a few strings into it, at runtime we're only able to see it as a Set.

14. Ce sunt tipurile de date algebrice?

Similar grupurilor din algebra - un set inchis de tipuri si functiii

functiile pot folosi doar anumite tipuri(cu ceil)

le folosim cand dorim restrictii mai bune

sunt mai dure deoarece avem un set de tipuri composite impreuna cu functiile necesare procesarii lor

15. Ce sunt clasele "sealed"?

protected la nivel de modul

Clasa a carei functionalitate e restransa la fisierul in care e descrisa.

- contine operatiile asociate

Sealed classes and interfaces represent restricted class hierarchies that provide more control over inheritance. All subclasses of a sealed class are known at compile time. No other subclasses may appear after a module with the sealed class is compiled. For example, third-party clients can't extend your sealed class in their code. Thus, each instance of a sealed class has a type from a limited set that is known when this class is compiled.

16. Ce sunt colectiile?

Niste entitati ce permit tratarea unitara a mai multor elemente

ca o multime matematica.

17. Ce operatii ne ofera colectiile lista?

add, remove, set, contains, get, isEmpty, size, indexOf

lastIndexOf, removeAt, clear()

18. Ce sunt listele in kotlin?

Colectii generice ordonate de elemente

19. Care e diferenta intre list si mutableList ?

Se pot adauga si sterge elemente

20. Ce sunt colectiile set?

O colectie de obiecte fara duplicate

21. Ce operatii ofera colectiile Set?

size, contains, add, remove

containsAll, isEmpty, retainAll

22. Ce este LinkedHashSet?

Implementarea unui set folosind un Hash_table

lista dubla inlantuita, hash table

Ma protejeaza de ordinea aleatorie a lui hashSet, pastreaza ordinea de la inserare

contains = O(1)

size, add = O(1)

23. Ce este un treeSet?

TreeSet este o interfata din SortedSet care foloseste un arbore ca metoda de stocare

log(n) - add, remove, contain

24. Ce este o colectie Map?

O colectie ce contine perechi de elemente

key -> value

suporta extragerea optimizata

cheile sunt unice

relatii 1 la 1

25. Ce operatii ne confera colectiile de tip Map?

containsValue, containsKey, isEmpty, size

keys, value, entries

put, remove, putAll

26. Cum se poate parcurge un map cu un for?

```
for elem in map.keys{
print("[%v]: %v", elem, map[elem])
}
```

27. Avantaje LinkedHashmap? Avantaje TreeMap?

LinkedHashMap = HashMap + LinkedList (lista dublu inlantuita)

TreeMap -> implementare pe arbori red-black

28. Ce este colectia Array?

Pentru Backward compatibility

E un container ce retine un nr fix de elemente de un tip dat

29. Care este relatia dintre colectiile Java si colectiile Kotlin?

Colectiile Kotlin is mai bune. Se distinge o data

dintre mutable si immutable

30. Ce sunt tipurile platforma?

Marcate cu !

Deseori librariile din Java(fiind un limbaj null-unsafe)

au metode ce returneaza un tip! deoarece Kotlin nu-si da

seama daca rezultatul e nullable sau nu.

Practic ! spune ca poate fi si nullable si non-nullable, venind dintr-o

platforma care nu are aceste clasificari

Se foloseste in string-uri, punem de exemplu

"%d". format(x)

31. Cand nu este imutabilul garantat in Kotlin?

In kotlin, e garantat de o interfata, dar Kotlin relaxeaza principiile

fata de java, deci pot aparea probleme la combinarea celor 2 limbaje.

Practic, atunci cand interoperam cu java(tipuri cochilie).

Mutable - se poate modifica

Immutable - nu se modifica, creeaza o noua colectie cu update-ul facut

Intrebari Curs8:

1) Ce este un model de proiectare (un design pattern)?

R: un model propus spre imitare, o metoda de a rezolva o anumita problema des intalnita

2) Cum a aparut termenul de model de proiectare (termenul de design pattern)?

R: de la un arhitect care avea de proiectat o casa

Cristopher Alexander

3) Cum a evoluat conceptul de design pattern?

R: in 1987 cunning si beck au dezvoltat un limbaj orientat, in 1990 gang of four a dezvoltat prima forma de catalog ce continea solutii pentru probleme particulare, 1995 prima carte cu toate explicatiile

4) Explicati modelele Real si Zuligoven ??? (not sure)

R: -model conceptual:descrie in termeni si concepte domeniul aplicatiei

-model de proiectare:descrie proiectarea software folosind constructii software

-model de programare:se fol. constructii de limbaj pentru a descrie forme

5) Unde sunt utile modelele de proiectare?

R: aplicatii complexe, pentru a reutiliza cod

OOP + ADT

6) Care sunt elementele unui model de proiectare?

R: -numele:trb. ales a.i. sa descrie pe scurt problema de proiectare

-problema:se descriu cazurile in care se aplica acest model

-solutia:descrie elem. care compun proiectul, relatiile si colaborarile dintre ele

-consecintele:rez. obtinute si compromisurile care trb facute cand se aplica modelul

7) Ce este o arhitectura inchisa?

R: o arhitectura care nu accepta modificari din exterior

8) Ce este o arhitectura deschisa?

R: una care poate fi modificata din exterior

9) Ce sunt modelele creationale?

R: se refera la abstractizarea instantierii unui obiect

incapsuleaza cunostintele despre clasele concrete

ascunde maniera in care clasele sunt create si cooperarea lor

10) Explicati fabrica de obiecte (factory method).

R: returneaza un obiect de tip produs, instantiaza un obiect

11) Cand se utilizeaza fabrica de obiecte?

R: Clasa nu poate anticipa ce tipuri de obiecte trebuie generate

12) Explicati fabrica de fabrici de obiecte.

R: este adaugat inca un nivel de abstractizare la fabrica de obiecte, mai multe categorii de fabrici

13) Ce este singleton si unde se utilizeaza?

R: cand dorim sa existe un singur obiect instantiat la un moment dat

exemplu manager la baza de date, un singur manager pt a evita concurenta

14) Explicati modelul Builder.

R: creeaza obiecte complexe pornind de la obiecte simple intr-o maniera incrementara

15) Cand se utilizeaza modelul Builder?

R: Obiectul nu poate fi creat intr-un singur pas

Evitarea crearii de prea multi constructori pt acelasi obiect

16) Explicati modelul Prototype.

R: Prototype e un blueprint pe baza caruia se creeaza alte obiecte.

17) Cand se utilizeaza modelul Prototip?

R: modelul creeaza o copie pornind de la un obiect existent atunci cand creearea de la zero este costisitoare din punct de vedere al resurselor

18) Ce sunt modelele structurale?

R: modele care imbina eficient mai multe clase sau obiecte in structuri care sa fie flexibile si eficiente

19) Explicati modelul Adapter.

R: face doua interfete care erau incompatibile sa fie compatibile

20) Modelul pod

R: este separata implementarea de abstractizare pentru a putea fi modificate separat

21)explicati modelul compus(composite)

R: este un arbore

putem sa tratam o lista de obiecte ca fiind un singur obiect

22)explicati modelul fatada

R: abstractieaza tot ce e in spate

furnizeaza o interfata pentru un sistem inalt, interfata mai usor de utilizat, permite o arhitectura in sistem

23)explicati modelul decorator

R: un model prin care putem adauga functionalitati noi la o clasa deja existenta fara a face modificari in clasa initiala

24)cand utilizam modelul decorator

R: dorim sa pastram clasa intacta, dar aducem noi functionalitati

25)modele comportamentale?

R: In software engineering, behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication.

26)ce este modelul intermediar(proxy)

R: instantiem in proxy un placeholder la un obiect si controlam accesul la obiectul initial

27)cand utilizam modelul proxy?

R: You need to support resource-hungry objects, and you do not want to instantiate such objects unless and until they are actually requested by the client.

28) modelul mediator?

R: This pattern provides a mediator class which normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling. Mediator pattern falls under behavioral pattern category.

Cand il folosim?

In case we have many objects that interact with each other, we end up with very complex and non-maintainable code. That’s why it is a good case to use the Mediator design pattern.

29)modelul strategie?

R: In the Strategy pattern, a class behavior or its algorithm can be changed at run time.

In the Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.

30)modelul vizitator?

The Visitor pattern is used when we have to perform an operation on a group of similar kind of Objects.
With the help of the visitor pattern, we can move the operational logic from the objects to another class.

- Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.
- The classic technique for recovering lost type information.

31)modelul memento?

- Without violating encapsulation, capture and externalize an object's internal state so that the object can be returned to this state later.
- A magic cookie that encapsulates a "check point" capability.
- Promote undo or rollback to full object status.

Cand il folosim?
Need to restore an object back to its previous state (e.g. "undo" or "rollback" operations).

32)modelul automat cu nr finit de stari?

The State design pattern is used when an Object changes its behavior based on its internal state.
If we have to change the behavior of an object based on its state, we can have a state variable in the Object and use if-else condition block to perform different actions based on the state.

33)modelul observator?

An Observer design pattern is useful when you are interested in the state of an object and want to get notified whenever there is any change.

34)model lant de responsabilitati?

Chain of responsibility pattern is used to achieve loose coupling in software design where a request from the client is passed to a chain of objects to process them. Later, the object in the chain will decide themselves who will be processing the request and whether the request is required to be sent to the next object in the chain or not.

35)modelul comanda?

The Command design pattern is used to encapsulate a request as an object and pass to an invoker, wherein the invoker does not know how to service the request but uses the encapsulated command to perform an action.

Intrebari Curs9:

TutorialsPoint

1) Explicati modelul Bridge.

R: Modelul Bridge este un model structural care iti permite

sa separi o clasa mare in doua ierarhii separate, abstractizare si implementare

care pot fi dezvoltate independent una de cealalta

Practic decuplam abstractizarea de implementare.

2) Cand utilizam modelul Bridge?

R: Cand avem ierarhii preferam sa folosim acest model

3) Explicati modelul Composite.

R: Modelul composite este un model de partitionare si descrie

un grup de obiecte care sunt tratate ca si o singura instanta

a unui acelasi tip de obiect

4) Cand utilizam modelul Composite?

R: Cand vedem ca mai multe tipuri de obiecte sunt tratate in acelasi fel, repetandu-se codul

pentru fiecare dintre ele, atunci e o idee buna sa folosim modelul composite, tratandu-le pe toate omogen.

6) Cand utilizam modelul Facade?

R: Cand construiesti o biblioteca (de ex), vrei sa ii oferi clientului "functiile" pe care le folosesti.

- Nu vrei sa-i arati functiile pe care le folosesti doar tu, intrinsec. Il arati doar ce poate reutiliza el.

7) Explicati modelul Decorator.

R: Modelul Decorator este un model structural ce permite adaugarea de noi functionalitati

unei clase, fara a-i afecta functionalitatile anterioare si modul cum relationeaza cu alte clase.

8) Cand utilizam modelul Decorator?

R: Ex. un context Manager (in python, context manager poate fi folosit ca un decorator)

9) Explicati modelul Proxy. (proxy = un intermediar)

R: ((Intermediarul proceseaza cererea noastra si o modifica sub o anumita forma pt a ne oferi rezultatul))

- Practic, proxy proceseaza o anumita cerere, un task pe care i-l dam.

10) Cand folosim Proxy?

R: Ex aplicatii de control parental: copilul vrea sa intre pe un URL (e trimis URL-ul ca si cerere unui proxy), iar daca

acesta este permis, atunci copilul va putea intra pe site. Daca nu, va primi un mesaj "Nu ai voie!".

11) Explicati modelul Flyweight.

R: Flyweight e un model de proiectare structural care ne permite sa salvam mai multe obiecte in

memorie prin pastrarea unor parti comune mai multor obiecte in loc sa punem obiectele intregi.

12) Ce este un model comportamental? (Behavioral)

R: Se ocupa de descrierea modului de interactiune intre clase

Descrie fluxul de control al unei aplicatii

13) Explicati modelul Chain of Responsibility.

R: Clientul trimite o cerere, iar aceasta este trimisa de la clasa la clasa pana se gaseste una care o poate procesa.

14) Cand utilizam un lant de responsabilitati?

R: De exemplu cand dorim tratarea unei cereri intr-o anumita ordine.

De exemplu, daca avem un request pe web, la tratarea lui il putem trece prin mai multe middleware-uri

de exemplu un middleware de securitate, care sa verifice daca persoana este logata, iar abia apoi ii putem trimite informatiile despre contul bancar, de exemplu

15) Explicati modelul Observer.

R: Observer este un model de comportament in care avem un Subject si mai multi observatori,

iar cand se schimba ceva intr-o instanta a unui obiect de tip Subject toti observatorii lui trebuie anuntati, fiinda acestia depind de subject

16) Cand folosim modelul Observer?

R: Cand avem o relatie one to many iar componentele many depind de componenta principala.

17) Explicati modelul automatului finit.

R: Are la baza modelul State.

18) Explicati modelul Visitor.

R: Visitor e un model de proiectare comportamental care ne permite sa separam algoritmii de obiectele asupra carora opereaza.

19) Explicati modelul Command.

R: Gestioneaza realizarea unei actiuni

20) Explicati modelul Memento.

R: Undo/redo

21) Explicati modelul Iterator.

R: Gestioneaza parcurgerea unei colectii de elemente.

22) Explicati modelul Strategy.

R: Se schimba obiectul in fct de strategie abordata (Ex. o parcare cu plata care are pret diferit de stationare pt zi si pt noapte)

23) Explicati modelul Mediator.

R: Un model care are ca scop reducerea complexitatii.

Intrebari Curs10:

1) Ce este calculul paralel?

R: Permite executarea mai multor programe in acelasi timp (desfasurarea simultana a mai multor procese)

- nr programe/procese = nr procesoare (avand in vedere ca ale noastre computere au maxim 8 procesoare, nu prea se aplica paralelismul, ci concurenta)

2) Ce este concurenta? (pseudoparalelism)

R: Concurenta = paralelism simulat (practic, mai multe procese au nevoie de o resursa comuna si concureaza pt a obtine acces la ea)

- nr de entitati care efectueaza ceva > nr procesoare

3) Ce este concurenta la nivel de date?

R: Se folosesc date (variabile) comune.

- Pot aparea probleme de coerenta: Daca un thread scrie o noua valoare intr-o variabila, iar altul citeste variabila respectiva, nu se stie sigur care valoare va fi citita (cea veche sau cea noua)

4) Ce este paralelismul la nivelul datelor?

R: Nu exista probleme in asigurarea coerentei. Se folosesc seturi de date separate (sau se impart datele initiale in bucati si se lucreaza separat -> fiecare proces/thread cu bucata lui)

5) Ce este o corutina?

R: Ca si concept, o corutina este similara unui thread, adica poate contine un bloc de instructiuni care se executa concurent cu restul codului. Totusi,

corutinele sunt mai light-weight si nu sunt legate de un anumit thread. Ele pot porni in cadrul unui thread si se pot termina in altul.

6) Explicati ciclul de viata al unei corutine.

R: stored -> on-stack -> running -> finished

- stored -> start minimal sub forma de obiecte copiate pe stiva

- on-stack -> datele corutinei au fost introduse in stiva de lucru, dar corutina nu a fost lansata in executie

- running -> o corutina intra in executie

- finished -> unde se ajunge cu terminarea corutinei respective

7) La ce se refera distrugerea la ordin?

R: oprirea fortata atunci cand se depaseste o anumita perioada de timp

8) Ce face cuvantul cheie "suspend"?

R: Cuvantul cheie suspend practic ne spune ca acea functie poate fi intrerupta oricand, dar poate fi si reluata oricand.

Lucrul asta este util daca vrem sa facem o planificare de genul Round - Robin

9) Ce face instructiunea "run blocking"?

R: Incepe o noua corutina si cumva desparte tot ce este in afara corutinei si ce este in interiorul corutinei. Practic blocheaza thread-ul care executa codul

din afara acestei corutine si il deblocheaza cand s-a terminat de facut tot ce era in interiorul corutinei

10) Ce scope-uri exista pt corutine?

R: Global, LifeCycle, ViewModel

11) Ce este un thread local?

R: Thread Local este o clasa ce se foloseste la a declara variabile ce se pot citi sau scrie de acelasi thread. De exemplu, avem 2 thread-uri care acceseaza acelasi cod care contine o referinta spre o variabila de tip thread local, atunci niciunul din thread-uri nu va vedea modificarile facute de celalalt thread la acea variabila.

12) Cum se poate asigura coerenta (integritatea) datelor?

R: Prin sincronizarea corutinelor, prin lock-uri, prin variabile atomice

13) Ce face instructiunea "async"?

R: Incepe o noua corutina, returneaza o valoare de tip Deffered, cumva promite ca va primi rezultatul mai tarziu. Folosim await pentru a-i primi valoarea unei variabile de tip Deffered

14) Cum se poate crea un thread in Kotlin?

R: Mostenind clasa Thread sau implementand interfata Runnable sau apeland functia thread() din kotlin

15) Ce tipuri de dispatcheri exista?

R: Main Dispatcher, IO Dispatcher, Default Dispatcher, Unconfined Dispatcher

16) La ce se refera reflexia si adnotarea in Kotlin?

R: *Reflection* is a set of language and library features that allows for introspecting the structure of your own program at runtime. Kotlin makes functions and properties first-class citizens in the language, and introspecting them (for example, learning a name or a type of a property or function at runtime) is closely intertwined with simply using a functional or reactive style.

Annotations are means of attaching metadata to code. To declare an annotation, put the annotation modifier in front of a class:

17) Ce este excluziunea mutuala?

R: Se refera la faptul ca la un moment dat de timp doar o corutina executa o bucata de cod. In Kotlin excluziunea mutuala se implementeaza in asa fel incat

nu sunt blocante: adica daca un proces are acces acum la "lacat" pt o bucata de cod atunci el asteapta ca alte procese sa-si faca treaba iar el va primi acel

"lacat" cand va fi liber ca apoi sa ruleze bucata lui de cod

18) Ce este interblocarea (deadlock)?

R: Un proces asteapta ceva ce a blocat (ocupat, folosit) celalalt si viceversa.

(LIVELOCK -> problema filosofilor -> spre deosebire de deadlock, nu se blocheaza, ci se incearca mereu)

19) Diferenta intre thread-uri simple si thread-uri locale.

R:

20) Metode de wait si notify.

R: Wait - opreste un thread pana functia notify ii spune sa revina in executie

Notify - trezeste din somn un anumit thread

21) Memorizarea (mecanism de caching) in contextul paralelismului.

R: **Memoization** is a technique to cache results of pure functions. A memoized function behaves as a normal function, but stores the result of previous computations associated with the parameters supplied to produce that result.

22) either?

R: An Either Monad consists of two basic types, Left and Right. It is, essentially, the If/Else statement of the Monad world. Left signifies that something went wrong, while Right signifies that things are working as they should. Just remember, that you want your code to be all right. As the monad passes through different bindings and is worked with, it can either report a Left, and, essentially, skip the rest of the call chain, or it can report Right and continue in fashion.

23) Modalitati de sincronizare

R: Variabile atomice, lock-uri, actori

24) Care e diferenta intre memoria comuna si cea cu transfer de mesaje?

R: In memoria cu transfer de mesaje, datele sunt transmise catre thread-uri prin intermediul unor cozi pt a evita aparitia problemelor

R: In cazul memoriei comune fiecare thread ia datele din acelasi loc din memorie, pot aparea probleme

25) Relatia intre corutine si threaduri.

R: Corutinele folosesc mult mai putine resurse si sunt foarte lightweight in comparatie cu thread-urile (putem lansa si 100k corutine fara sa avem probleme)

26) Cum pot mapa corutine pe threaduri?

R: Corutinele isi pot porni executia pe un thread si pot termina pe orice altul

27) Relatia intre corutine si JVM?

R: Corutinele pot fi lansate in cadrul unui thread si sunt active atat timp cat thread-ul este activ.

28) Ce este un actor? (in contextul corutinelor)

R: Un actor e format din 3 lucruri (o corutina, un canal de comunicatie si o stare interna)

29) explicati granularitatea corutinelor

se refera la caracteristicile ce definesc o stare

gran. Mare (la nivel de cod), defineste un task

gran. medie (la nivel de control), defineste functii

gran. Fina(nivel date)

gran. Foarte fina(alegeri multiple) cu suport hard

30)cum se poate forta terminarea unei corutine dupa o cuanta de timp

comanda cancel, comanda delay inainte eventual

31)programare asincrona

exemplu server, avem operatii ce nu necesita numite date

32)cum se poate crea un thread in kotlin

ca in java thread and runnable

33)monada

Think of a Monad as a box. In this box, you can store stuff. A Monad is simply a data wrapper with a few different, unifying properties.

34)alias de tip

Type aliases provide alternative names for existing types. If the type name is too long you can introduce a different shorter name and use the new one instead.

Intrebari Curs 11:

1) python GIL?

The Python Global Interpreter Lock or GIL, in simple words, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter.

This means that only one thread can be in a state of execution at any point in time. The impact of the GIL isn't visible to developers who execute single-

threaded programs, but it can be a performance bottleneck in CPU-bound and multi-threaded code.

2) diferenta dintre lock si rlock

Locks	RLocks
A Lock object can not be acquired again by any thread unless it is released by the thread which which is accessing the shared resource.	An RLock object can be acquired numerous times by any thread.
A Lock object can be released by any thread.	An RLock object can only be released by the thread which acquired it.
A Lock object can be owned by none	An RLock object can be owned by many threads
Execution of a Lock object is faster.	Execution of an RLock object is slower than a Lock object

3) explicati semaforul

un tip de data abstract, asemanator cu lock ul doar ca poate sa functioneze pe mai multe thread uri

4) ce este un fir cu conditie

un thread care se executa doar daca este adevarata o anumita conditie pe care o stabilim la inceput

-ce este conditia?

este o variabila care permite thread urilor sa astepte pana cand este anuntat de alt thread

-ce are in spate un obiect de tip condition?

un lock, un mutex

5)cum utilizam with in cazul thread urilor

Aquire si release automat

6)cum utilizam cozile la comunicarea intre thread uri

pot fi folosite la problema producator consumator

7)ce ne ofera modulul multiprocessing

paralelism, paralelism real putem avea paralelism doar la nivel de procese

poate utiliza procesorul pentru a executa mai multe task uri

8)diferenta dintr thread uri si procese in python

un proces contine mai multe thread uri

9)cand folosim un proces sau un thread

thread-taskuri de intrare iesire pentru a nu bloca aplicatia si nu folosim un proces

10)cum este creat un proces prin mostenire

extindem clasa thread si punem clasele intr-un proces

un thread pool

11)cum utilizam pipeurile in comunicarea intre procese

iesirea este intrare pentru un alt proces

12)ce este un pipe?

o structura de date cu un capat de citire si unul de scriere sau ce era mai sus

13)ce este o bariera?

un obiect care asteapt un numar de thread uri sa se opreasca pentru a permite unui nou thread sa inceapa

14)cum utilizam un pool de procese si ce are in spate?

```
def test_process_pool_executor(function, list_of_integers):
    with ProcessPoolExecutor(cpu_count()) as pool:
        return pool.map(function, list_of_integers)
```

Functia fork plus o coada pt task uri in spate?

15)ce este un executor din python

The **executor** package is a simple wrapper for **Python's** subprocess module that makes it very easy to handle subprocesses on UNIX systems with proper escaping of arguments and error checking: An object oriented interface is used to execute commands using sane but customizable (and well documented) defaults.

16)ce este o aplicatie simpla client-server

In principle, a **client/server application** consists of a **client** program that consumes services provided by a **server** program. The **client** requests services from the **server** by calling functions in the **server application**.

Intrebari Curs 12:

1.Dati exemple de transformari intre spatii

The Kotlin standard library provides a set of extension functions for collection *transformations*. These functions build new collections from existing ones based on the transformation rules provided.

2. Ce este banda "moebils"?

Este un model de suprafață cu o singură față și o singură margine.

Banda are proprietatea matematică de a fi neorientabilă

3. Care este ipoteza lui Church?

Fiecare functie care poate fi calculata poate fi scrisa recursiv

4. Care e ipoteza lui Turing?

5. Ce este o functie anonima?

Este o functie fara nume

6. Ce e lambda?

Functii pe care le putem crea fara a le asigna un nume explicit

7. Care sunt limitarile Java in cazul calculului functional?

In Java, you must target every lambda expression to a specific functional interface type. This is by design, and in practice it means that functions are not first-class citizens of the language, spre deosebire de kotlin.

8. Cum se poate utiliza o functie ca o proprietate?

Utilizarea unei funcții ca o proprietate

```
fun main(args: Array<String>)
{
   val dif_numere = { x: Int, y: Int -> x - y }
   println("Diferenta 1 este ${dif_numere(33,66)}")
   println("Diderenta 2 este ${dif_numere(77,11)}")
}
```

9. Ce tip de date au functiile lambda in kotlin?

Lambdas Expressions are essentially anonymous functions that we can treat as values – we can, for example, pass them as arguments to methods, return them, or do any other thing we could do with a normal object.

10. Dati un exemplu de o functie de nivel superior in kotlin

O functie care accepta parametri si poate returna o alta functie

fold

11. Ce este un efect lateral?

Este acel efect ce consta in modificarea variabilelor existente

in exteriorul blocului functii

12. Ce este o functie pura?

O functie care nu are efecte laterale(nu schimba variabilele din exteriorul ei)

13. Ce face cuvantul cheie vararg?

Permite primirea unui numar variabil de argumente

14. Explicati parametrii alias

Ca un typedef, redenumim tipuri existente

15. Ce este o functie extensie?

Ce este o functie ce extinde o clasa, ca si cum am face o noua clasa la care mai adaugam o functie

16. Care e diferenta dintre functia unei clase, functia supraincarcata intr-o clasa derivata

si functia extensie acelei clase?

Inheritance is nothing but extending a class and adding functionality to it.

Extension functions are the functions where you add a functionality to an existing class by just adding a function without extending the class and this will be resolved statically and all the objects of that class can use it.

17. Ce este un dispatcher receiver?

The instance of the class in which the extension is declared is called ***dispatch receiver***, and the instance of the receiver type of the extension method is called ***extension receiver***.

Dispatch recv

```
open class Felina
open class Pisica():Felina()
open class Primata(val name: String)
fun Primata.vorbeste() = "$name: face uhaha uhaha"//extensie primate
open class MaimutaMare(name: String) : Primata(name)
fun MaimutaMare.vorbeste() = "${this.name} : urlet" //ignorata deoarece amm deja un vorbeste
din primata
fun mesajScris(primata: Primata) {   println(primata.vorbeste()) }
open class Ingrijitor(val name: String) {
  open fun Felina.react() = "HRRMR!!!"
  fun Primata.react() = "$name se joaca cu ${this@Ingrijitor.name}" //ditribuitor intern bagat aici
pentru ca altfel nu am acces la .name
  fun areGrija(felina: Felina) {       println("Felina face: ${felina.react()}")    }
  fun areGrija(primata: Primata){   println("Primata spune: ${primata.react()}")     }
  fun Ingrijitor.react() = "$name din afara clasei se joaca cu ${this@Ingrijitor.name}" //neglijat }
open class Vet(name: String): Ingrijitor(name) {   override fun Felina.react() = "fuge de $name"   }
```

18. Cand exista posibilitatea unui conflict de nume in utilizarea functiilor de extensie?

when an extension function has the same name as a member function

19. Ce sunt functiile de extensie pt obiecte?

Funcții de extensie pentru obiecte

```
object Constructor {
}

fun Constructor.casaNouaCaramida() = "casa pe pamant"

class Proiectant {
   companion object {
   }
   object Birou {
   }
}
fun Proiectant.Companion.prototipNou() = "montaj test"
fun Proiectant.Birou.mapaDeLucrari() = listOf("Proiect casa", "Proiect bloc")
fun main()
{
   println(Constructor.casaNouaCaramida())
   println(Proiectant.prototipNou())
   Proiectant.Birou.mapaDeLucrari().forEach(::println)
}
```

20. Ce sunt functiile infix?

Kotlin allows some functions to be called without using the period and brackets. **These are called *infix* methods, and their use can result in code that looks much more like a natural language.**

21. La ce este buna functia map?

Returneaza o lista ce contine rezultatele aplicatii unei transformari

listei initiale

22. Cand utilizam functia filter?

Cand dorim sa selectam doar anumite elemente dintr-un container

23. Ce face functia flatMap?

Imi "aplatizeaza", de exemplu, mai multe liste intr-o lista.

Transforma dintr-o lista de liste intr-o lista

24. Ce rol au functiile drop si take?

Elimina un nr de elemente din lista sau selecteaza un nr de elemente din lista

25. Ce este functorul?

A functor is any class that holds data and can be mapped over.

26. Ce sunt functiile curry?

A curried function is a function that takes multiple arguments *one at a time.* Given a function with 3 parameters, the curried version will take one argument and return a function that takes the next argument, which returns a function that takes the third argument. The last function returns the result of applying the function to all of its arguments.

Intrebari Curs 13:

1) Ce este calculul Lambda?

R: Este un model matematic formal in care folosim functii cu parametri, fara nume, pt calculul unor expresii.

2) Cum s-ar putea rescrie un if cu un lambda utilizand doua obiecte?

R:

true = lambda x, y: x

false = lambda x,y: y

if = lambda p, x, y: p(x,y)

3) La ce se refera *args si **kwargs?

R: Permit introducerea unui nr variabil de argumente la o functie

args - tupla

kwargs - dictionar

4) La ce se refera functiile de prim nivel?

R: **First**-class **functions** means that the language treats **functions** as values – that you can assign a **function** into a variable, pass it around etc. It's seldom used when referring to a **function**, such as “a **first**-class **function**”.

5) La ce se refera datele imutabile?

R: Datele imutabile se refera la datele care nu-si pot modifica continutul

6) La ce se refera functiile pure?

R: Functiile pure sunt cele fara efecte laterale.

7) La ce se refera abordarea impacheteaza - proceseaza - despacheteaza?

R: Tuple Packing and Unpacking

When we access individual elements in tuples, we can use indexing because tuples support subscripts using indices

8) Cum se realizeaza evaluarea la cerere?

Evaluare la cerere

```
True and print("and cu true")
False and print("and cu false")
1 and print("and cu unu")
0 and print("and cu zero")
True or print("or cu true")
False or print("or cu false")
```

si rezultatul executiei

```
and cu true
and cu unu
or cu false
```

R:

9) Ce este un generator recursiv?

R: generator care se apeleaza singur

10) Pentru ce utilizam modulul iterTools?

R: Pt a folosi functii care genereaza si proceseaza serii si secvente de numere (pt partea hardware sunt f bune deoarece se pot genera repede si usor semnale)

11) Cand utilizam iteratorii infiniti?

R: Sunt utili pt generarea semnalelor (sinus, cosinus etc.)

12) Cum se calculeaza eroarea prin acumulare?

R: Se aduna putin cu putin.

13) Care-i diferenta intre functiile "cycle" si "accumulate"?

R: cycle -> repeta argumentele la infinit

accumulate -> efectueaza o operatie asupra fiecarui argument

14) Cum se poate utiliza modulul iterTools pt a crea functii de ordin 2?

itertools - creare funcții de ordin doi

- pentru reamintire: s(n) = p * s(n-1) + q * s(n-2) + r

```
import itertools as it
def secventaOrdinDoi(p, q, r, initial_values):
    intermediate = it.accumulate(
        it.repeat(initial_values),
        lambda s, _: (s[1], p*s[1] + q*s[0] + r) )
    return map(lambda x: x[0], intermediate)
numereFibonacci = secventaOrdinDoi(1, 1, 0, (0, 1))
secvNumereFibonacci = list(next(numereFibonacci) for _ in range(8))
print(secvNumereFibonacci)
numerePell = secventaOrdinDoi(2, 1, 0, (0, 1))
secvNumerePell = list(next(numerePell) for _ in range(6))
print(secvNumerePell)
numereLucas = secventaOrdinDoi(1, 1, 0, (2, 1))
secvNumereLucas = list(next(numereLucas) for _ in range(6))
print(secvNumereLucas)
numereFibonacciAlternative = secventaOrdinDoi(-1, 1, 0, (-1, 1))
secvNumereFibonacciAlternative = list(next(numereFibonacciAlternative) for _ in range(6))
print(secvNumereFibonacciAlternative)
```

si rezultatul executiei

```
[0, 1, 1, 2, 3, 5, 8, 13]
[0, 1, 2, 5, 12, 29]
[2, 1, 3, 4, 7, 11]
[-1, 1, -2, 3, -5, 8]

Process finished with exit code 0
```

R:

15) Cand se utilizeaza functie "tee"?

R: multiplica iteratori

16) Cand utilizam operatorii "any" si "all"?

R: any -> macar una dintre "ele" sa respecte o anumita conditie (conditii cu || intre ele) -> returneaza true daca macar o conditie e true

all -> toate dintre "ele" trb sa respecte o anumita conditie (conditii cu && intre ele) -> returneaza true daca toate conditiile sunt true

17) Explicati data flatten.

R: concatenare colectii

18) Ce este scurtcircuitul (nu ala electric)?

R: De ex intr-un if in care avem mai multe conditii cu "OR", daca prima cond e adevarata

atunci nu mai trb verificate si celelalte -> scurtcircuit

19) Cum putem evalua deciziile in calculul functional?

R:

20) Ce este un "closure"?

R: Functiile in python sunt referinte catre obiecte. Deci ele pot suporta si apeluri de alte obiecte???

-inchiderile in python retin si contextul in care au fost incheiate

21) Care operatori pe colectii sunt mai rapizi si in ce domeniu?

R:

22) Care este diferenta intre un decorator in stil macro si unul in stil OOP?

R: - stil macro (pe functie)

- still OOP (pe clasa)

23) Cum putem implementa un state machine (FSM)?

R: vezi curs 13 final