

SV108
Kol 101 - Să se realizeze un program Kotlin care va aplica memoizarea generalizată pentru calculul sirului $f(i)=f(i-1)+f(i-2)$ pentru $i=1..n$, utilizând un concurrent hashmap.

Py1334 - Utilizând modelul lant de responsabilități și nivelul de alertă [0..5] să se creeze un program Python care va distribui un mesaj (primit la intrare împreună cu nivelul lui) pentru a fi tratat(afişat) în zona potrivită (e.g. 5-paznici muzee, 4-politie, 3 sni, 2 sie, 1-csat, 0-nato). Se vor desena diagrame de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

problema1-KOTLIN

Sa se realizeze un program KOTLIN care va aplica memorizarea generalizata pentru calculul sirului $f(i)=f(i-1)+f(i-2)$ pentru $i=1...n$,utilizand un concurrent hashmap.

Problema 2-PYTHON

Utilizand modelul lant de responsabilitati si nivelul de alerta[0..5] sa se creeze un program Python care distribuie un mesaj (primit la intrare impreuna cu nivelul lui) pentru a fi tratat (afisat) in zona potrivita .Se vor desena giagrama de 4 clase si de obiecte.Se va explica maniera de aplicare a principiilor SOLID.

PROBLEMA 1 KOTLIN

```
import java.util.concurrent.ConcurrentHashMap  
  
import kotlin.system.measureNanoTime  
  
  
class SimpleFunctionWithMemoization(var m: ConcurrentHashMap<Int, Int>) {  
  
    init {  
        m[0] = 1  
    }  
}
```

```

m[-1] = 1
}

// Funcție cu memorizare
fun f(i: Int): Int {
    if (m[i] != null)
        return m[i]!! // Returnează valoarea memorizată dacă există

    if (i == 0 || i == -1)
        return 1 // Caz de bază: returnează 1 pentru i = 0 sau i = -1

    m.putIfAbsent(i, f(i - 1) + f(i - 2)) // Calculează recursiv valoarea și o memorează
    return m[i]!! // Returnează valoarea memorizată
}

class SimpleFunctionWithoutMemoization {

    // Funcție fără memorizare
    fun f(i: Int): Int {
        if (i == 0 || i == -1)
            return 1 // Caz de bază: returnează 1 pentru i = 0 sau i = -1
        return f(i - 1) + f(i - 2) // Calculează recursiv valoarea fără memorizare
    }
}

fun main() {
    val m = ConcurrentHashMap<Int, Int>()
    val a = SimpleFunctionWithMemoization(m) // Creează o instanță a clasei
    SimpleFunctionWithMemoization
}

```

```

val b = SimpleFunctionWithoutMemoization() // Creează o instanță a clasei
SimpleFunctionWithoutMemoization

var time = measureNanoTime { print(a.f(40)) } // Măsoară timpul de execuție al funcției cu memorizare
println(" cu timpul $time ns")

time = measureNanoTime { print(b.f(40)) } // Măsoară timpul de execuție al funcției fără memorizare
println(" cu timpul $time ns")

}

```

//////////
PROBLEMA 2 PYTHON

```
from abc import abstractmethod
```

```
# Definirea clasei de bază AbstractSecurity
```

```
class AbstractSecurity:
```

```
    MUSEUM_GUARDIAN = 5
```

```
    POLICE = 4
```

```
    SRI = 3
```

```
    SIE = 2
```

```
    CSAT = 1
```

```
    NATO = 0
```

```
_next = None # Referința către următorul element din lanțul de responsabilitate
```

```
_level: int
```

```
def set_next_in_chain(self, next_in_chain):
```

```
    self._next = next_in_chain
```

```

def process_message(self, level, message):
    if self._level <= level:
        self.process(message) # Procesează mesajul dacă nivelul este corespunzător
    else:
        self._next.process_message(level, message) # Trimite mesajul către următorul element din lanț

    @abstractmethod
    def process(self, message):
        pass

# Implementarea specifică a clasei AbstractSecurity pentru MuseumGuardian
class MuseumGuardian(AbstractSecurity):

    def __init__(self, level):
        self._level = level

    def process(self, message):
        print("I'm the guardian and I can do that: " + message)

# Implementarea specifică a clasei AbstractSecurity pentru Police
class Police(AbstractSecurity):

    def process(self, message):
        print("The police will solve that: " + message)

    def __init__(self, level):
        self._level = level

```

```
# Implementarea specifică a clasei AbstractSecurity pentru SRI  
class SRI(AbstractSecurity):
```

```
    def __init__(self, level):  
        self._level = level  
  
    def process(self, message):  
        print("We are the SRI, we take that: " + message)
```

```
# Implementarea specifică a clasei AbstractSecurity pentru SIE  
class SIE(AbstractSecurity):
```

```
    def __init__(self, level):  
        self._level = level  
  
    def process(self, message):  
        print("This is SIE: " + message)
```

```
# Implementarea specifică a clasei AbstractSecurity pentru CSAT  
class CSAT(AbstractSecurity):
```

```
    def __init__(self, level):  
        self._level = level  
  
    def process(self, message):  
        print("CSAT: " + message)
```

```
# Implementarea specifică a clasei AbstractSecurity pentru NATO  
class NATO(AbstractSecurity):
```

```
def __init__(self, level):
    self._level = level

def process(self, message):
    print("NATO: " + message)

# Clasa SecurityChain
class SecurityChain:

    @staticmethod
    def get_security_chain() -> AbstractSecurity:
        guardian = MuseumGuardian(AbstractSecurity.MUSEUM_GUARDIAN)
        police = Police(AbstractSecurity.POLICE)
        sri = SRI(AbstractSecurity.SRI)
        sie = SIE(AbstractSecurity.SIE)
        csat = CSAT(AbstractSecurity.CSAT)
        nato = NATO(AbstractSecurity.NATO)

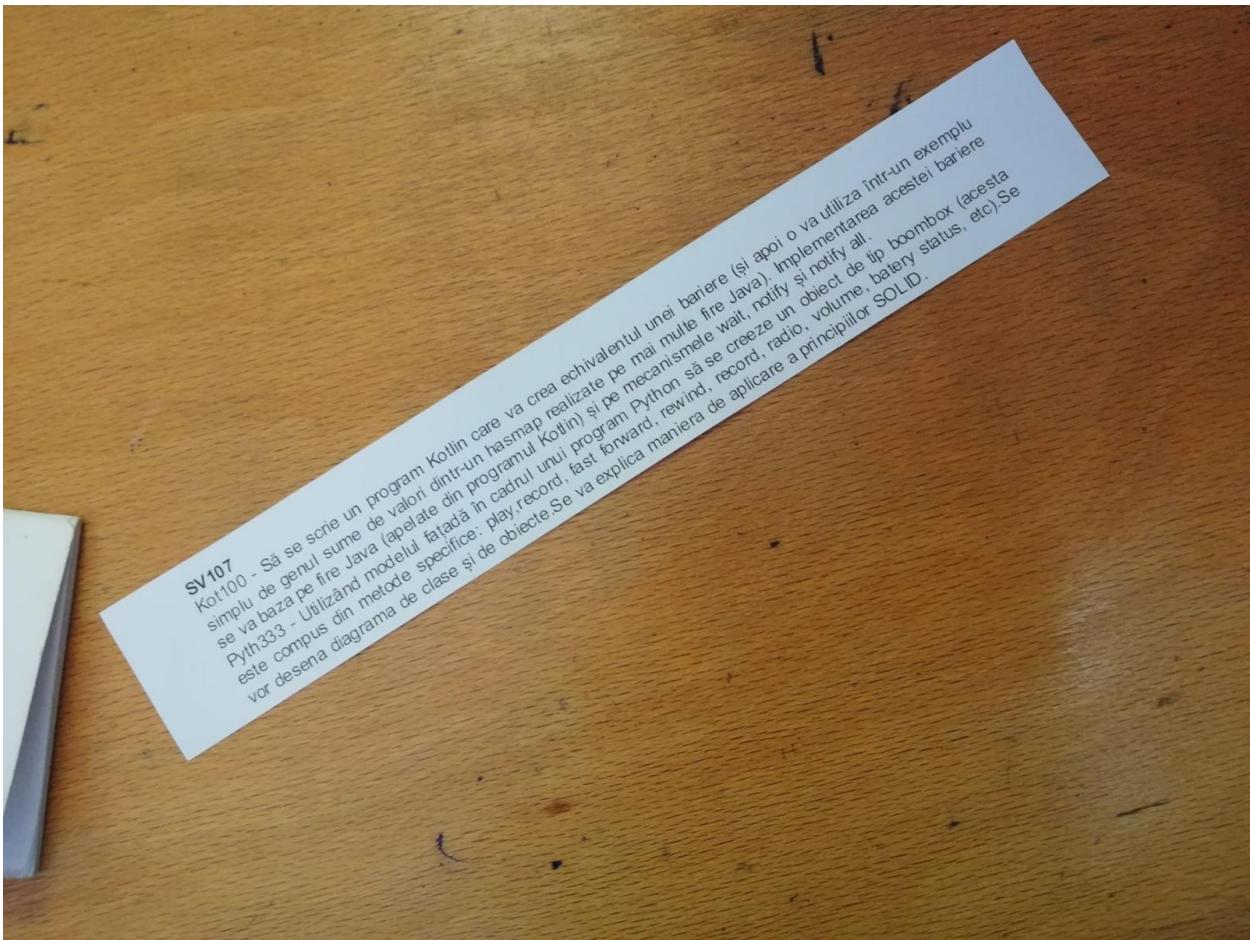
        guardian.set_next_in_chain(police)
        police.set_next_in_chain(sri)
        sri.set_next_in_chain(sie)
        sie.set_next_in_chain(csat)
        csat.set_next_in_chain(nato)

    return guardian

# Punctul de intrare în program
if __name__ == '__main__':
```

```
security_chain = SecurityChain.get_security_chain()
```

```
security_chain.process_message(4, "Get the thief!") # Mesajul va fi procesat de Police
```



SV107

KOT100

Sa se scrie un program Kotlin care va crea echivalentul unei bariere (si apoi va utiliza intr un exemplu simplu de genul sume valori dintr un hashmap realizate pe mai multe fire Java).Implementarea acestei bariere se va baza pe fire JAVA si pe mecanismele wait,notify si notify all.

PYTH333

Utilizand modelul fatatda in cadrul unui program Python sa se creeze un obiect de tip boombox (acesta este compus din metode specifice:play,record,fast forward, redwind, record,radio,volume,battery status, etc).Se vor desena diagrama de clase si de obiecte.Se va explica maniera de aplicare a principiilor SOLID.

PROBLEMA 2 PYTHON

```
import abc

# Definirea clasei BoomboxFacade
class BoomboxFacade:
    def __init__(self):
        self.boombox = Boombox()

    def Play(self):
        if self.boombox.mode.GetMode() == "On tape mode": # Verifică modul de funcționare al boombox-ului
            if self.boombox.tape != None: # Verifică dacă există o casetă introdusă
                self.boombox.tape.Play() # Rulează caseta
                self.boombox.mode = OnTapeMode() # Setează modul de funcționare pe modul casetă
            else:
                print("Please Insert Tape") # Afisează un mesaj de eroare dacă nu există casetă
        else:
            print("Radio Started") # Afisează un mesaj dacă boombox-ul rulează în modul radio

    def Stop(self):
```

```
if self.boombox.mode.GetMode() == "On tape mode":  
    if self.boombox.tape != None:  
        self.boombox.tape.Stop() # Oprește caseta  
    else:  
        print("No Tape Inserted") # Afisează un mesaj de eroare dacă nu există casetă  
else:  
    print("Radio Stopped") # Afisează un mesaj dacă boombox-ul rulează în modul radio  
  
# Metode pentru controlul casetei  
  
def FastForward(self):  
    if self.boombox.tape != None:  
        self.boombox.tape.FastForward() # Avans rapid pe casetă  
    else:  
        print("No Tape Inserted") # Afisează un mesaj de eroare dacă nu există casetă  
  
def Rewind(self):  
    if self.boombox.tape != None:  
        self.boombox.tape.Rewind() # Derulare înapoi pe casetă  
    else:  
        print("No Tape Inserted") # Afisează un mesaj de eroare dacă nu există casetă  
  
def InsertTape(self, tape):  
    self.boombox.InsertTape(tape) # Introduce o casetă în boombox  
  
def RemoveTape(self):  
    self.boombox.RemoveTape() # Scoate caseta din boombox  
  
# Metode pentru afişarea stării boombox-ului  
def ShowBattery(self):
```

```
print(str(self.boombox.GetBattery()) + "% left") # Afisează nivelul bateriei

def ShowStatus(self):
    print("Current status: " + self.boombox.GetStatus()) # Afisează starea curentă a boombox-ului

# Metode pentru controlul volumului
def VolumeUp(self):
    self.boombox.volumeManager.VolumeUp() # Crește volumul

def VolumeDown(self):
    self.boombox.volumeManager.VolumeDown() # Scade volumul

def ShowVolume(self):
    print("Volume: " + str(self.boombox.volumeManager.GetVolume())) # Afisează volumul curent

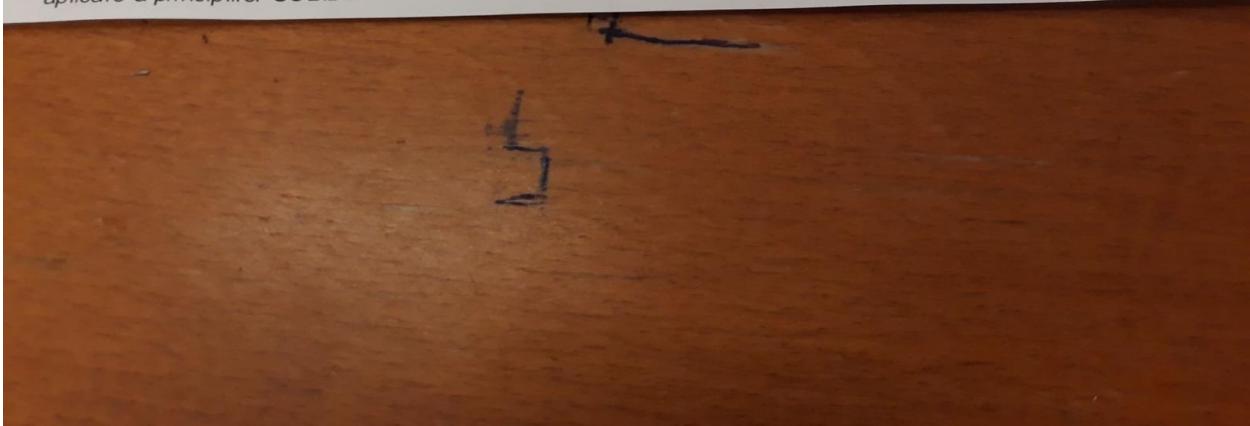
# Metode pentru înregistrare
def Record(self):
    if self.boombox.tape != None:
        self.boombox.recordingManager
```



SV88

Kot91 - Să se scrie un program Kotlin care va utiliza subclase active (cu mutex și lock) pentru procesarea simultană unui hasmap bazată. Clasa de bază va stabili operațiile (de ex adunare, scădere înmulțire împărțire) iar subclaselor vor pune la dispoziție obiecte care se execută în fire separate. Se va prezenta diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

Py308 - Utilizând fabrica de fabrici să se creeze un program Python care va afișa în funcție de apel în mod grafic un pătrat un dreptunghi sau un cerc. Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.



SV88

Kot91

Sa se scrie un program Kotlin care va ultiza subclase active (cu mutex si lock) pentru procesarea simultanata unui hashmap bazata. Clasa de baza va stabili operatiile (de ex adunare, scadere, inmultire, impartire) iar subclasele vor pune la dispozitie obiecte care se executa in fire separate. Se va prezenta diagrama de clase si de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

PY308

Utilizand fabrica de fabrici sa se creeze un program Python caree va afisa in functie de apel in mod grafic un patrat un dreptunghi sau un cerc.Se vor desena diagrama de clase si de obiecte.Se va explica maniera de aplicare a principiilor SOLID.

PROBLEMA 2 PYTHON

```
import abc

from abc import ABC

import tkinter as tk

import math

# Definim clasa abstractă de bază pentru forme

class IShape(metaclass=abc.ABCMeta):

    @abc.abstractmethod

    def drawShape(self, xAxis, yAxis, canvas):

        pass

# Definim clasa abstractă de bază pentru forme rotunjite

class IRoundedShape(IShape, ABC):

    def __init__(self, radius):

        self.radius = radius

# Definim clasa abstractă de bază pentru forme dreptunghiulare

class IRectangleShape(IShape, ABC):

    def __init__(self, width, height):

        self.width = width
```

```

        self.height = height

# Definim clasa Circle (Cerc)
class Circle(IRoundedShape):
    def __init__(self, radius):
        super().__init__(radius)

    def drawShape(self, xAxis, yAxis, canvas):
        print("Desenăm un cerc cu rază " + str(self.radius))
        if isinstance(canvas, tk.Canvas):
            canvas.create_oval(xAxis, yAxis, xAxis + 2*self.radius, yAxis + 2*self.radius, fill="red")

# Definim clasa Rectangle (Dreptunghi)
class Rectangle(IRectangleShape):
    def __init__(self, width, height):
        super().__init__(width, height)

    def drawShape(self, xAxis, yAxis, canvas):
        print("Desenăm un dreptunghi cu lățimea " + str(self.width) + " și înălțimea " + str(self.height))
        if isinstance(canvas, tk.Canvas):
            canvas.create_rectangle(xAxis, yAxis, xAxis + self.width, yAxis + self.height, fill="blue")

# Definim clasa Square (Pătrat)
class Square(IRectangleShape):
    def __init__(self, length):
        super().__init__(length, length)

    def drawShape(self, xAxis, yAxis, canvas):
        print("Desenăm un pătrat cu latura " + str(self.width))

```

```

if isinstance(canvas, tk.Canvas):
    canvas.create_rectangle(xAxis, yAxis, xAxis + self.width, yAxis + self.width, fill="green")

# Definim clasa abstractă de bază pentru fabrici de forme

class IFactory(metaclass=abc.ABCMeta):

    @abc.abstractmethod
    def createShape(self, shapeType) -> IShape:
        pass

# Definim clasa RoundShapesFactory (Fabrică pentru forme rotunjite)

class RoundShapesFactory(IFactory):

    def __init__(self):
        self.radius = 10

    def createShape(self, shapeType) -> IShape:
        if shapeType == "Circle":
            return Circle(self.radius)
        raise Exception("Obiect necunoscut")

    @property
    def radius(self):
        return self.radius__

    @radius.setter
    def radius(self, radius):
        if radius <= 0:
            raise Exception("Raza trebuie să fie pozitivă!")
        self.radius__ = radius

```

```
# Definim clasa RectangleShapesFactory (Fabrică pentru forme dreptunghiulare)

class RectangleShapesFactory(IFactory):

    def __init__(self):
        self.width = 10
        self.height = 5

    def createShape(self, shapeType) -> IShape:
        if shapeType == "Rectangle":
            return Rectangle(self.width, self.height)
```

SV86
Kot 76 - Utilizând modelul observator să se creeze în pentru un obiect de tip recalculare a prețului, în funcție de niște rate de reducere introduse extern de la tastatură (prin intermediul unui model proxy cu validare pe user,parolă), un logger (scriere automată în jurnal a operațiilor) care va scrie într-un fișier separat user și data la care a fost efectuată modificarea de preț și ce modificare a fost realizată).Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.
Py306 - Utilizând fabrica de fabrici să se creeze un program Python care în funcție de limba selectată în apel să întoarcă un buton a cărui etichetă să fie în limba selectată la apel și apoi acesta să fie afișat pe ecran utilizând Tkinter. Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

SV86

Kot76

Utilizând modelul obseervator sa se creeze in pentru un obiect de tip recalculare a pretului in functie de niste rate de reducere introduse extern de la tastatura (prin intermediul unui model proxy cu validare pe user,parola),un logger(screierea automata in jurnal a operatiilor)care va scrie intr un fisier separat user si data la care a fost efectuata modificarea de pret si ce modificare a fost realizata).Se vor desena diagrama de clase si de obiecte.Se va explica maniera de aplicare a principiilor SOLID.

Py306

Utilizand fabrica de fabrici sa creeze un program Python care in functie de limba selectata in apel sa intoarca un buton a carui eticheta sa fie in limba selectata la apel si apoi acesta sa fie4 afisat pe ecran

utilizand tkinter.Se vor desena diagrama de clase si de obiecte .Se va explica maniera de aplicare a principiilor SOLID.

PROBLEMA 1 KOTLIN

```
import java.io.IOException
import java.nio.file.Files
import java.nio.file.Paths
import java.nio.file.StandardOpenOption
import java.time.LocalDateTime
import kotlin.system.exitProcess

// Interfața Observer
interface Observer {
    fun update(realUser: RealUser)
}

// Interfața Observable
interface Observable {
    fun attach(o: Observer)
}

// Interfața User
interface User {
    fun login()
}
```

```
// Interfață Logger

interface Logger {
    fun writeData(data: String)
}

// Clasa abstractă Product

abstract class Product(private var price: Double) {
    protected var name: String = ""

    fun getPrice(): Double {
        return this.price
    }

    fun getProductName(): String {
        return this.name
    }

    fun setPrice(price: Double) {
        this.price = price
    }
}

// Clasa Tomatoes care extinde clasa Product

class Tomatoes(price: Double) : Product(price) {
    init {
        this.name = "Tomatoes"
    }
}
```

```
// Clasa PriceCalc care implementează interfața Observer

class PriceCalc(private val p: Product) : Observer {

    private val logger = MyLogger("logger.txt")

    override fun update(realUser: RealUser) {
        // Citim rata de discount de la tastatură
        print("Enter the discount rate: ")

        val per = KeyboardReader.read()?.toFloat()
        if (per != null) {
            if (per > 0 && per < 90) {
                // Calculăm prețul nou cu discount
                val new: Double = this.p.getPrice() - this.p.getPrice() * (per / 100)

                // Scriem informațiile în jurnal
                this.logger.writeData("Change made by " + realUser.getUsername() + "\n")
                this.logger.writeData("Date: " + LocalDateTime.now().toString() + "\n")
                this.logger.writeData("The price has changed for: " + this.p.getProductName() + "\n")
                this.logger.writeData("Old price: " + this.p.getPrice() + "lei/kg\n")
                this.logger.writeData("New price: " + new.toString() + "lei/kg\n")
                this.logger.writeData("Discount: $per %\n\n")

                // Setăm prețul nou al produsului
                p.setPrice(new)
            }
        }
    }
}
```

```
}
```

```
// Clasa KeyboardReader pentru citirea de la tastatură
```

```
class KeyboardReader {
```

```
    companion object {
```

```
        fun read(): String? {
```

```
            return readLine()
```

```
        }
```

```
    }
```

```
}
```

```
// Clasa ProxyUser care implementează interfața User
```

```
class ProxyUser(private val realUser: RealUser) : User {
```

```
    override fun login() {
```

```
        // Citim numele de utilizator și parola de la tastatură
```

```
        print("User: ")
```

```
        val user = KeyboardReader.read()
```

```
        print("Password: ")
```

```
        val pass = KeyboardReader.read()
```

```
        // Verificăm autentificarea utilizatorului
```

```
        if (!(user != null && realUser.getUsername() == user)) {
```

```
            print("Invalid username!")
```

```
            return
```

```
        }
```

```
        if (!(pass != null && pass == realUser.getPassword())) {
```

```
            print("Invalid password!")
```

```
    return
}

// Autentificare validă, se apelează metoda de login a utilizatorului real
realUser.login()

}

}

// Clasa RealUser care implementează interfața User și Observable
class RealUser(private val user: String, private val password: String) : User, Observable {

    private val viewers = mutableListOf<Observer>()

    override fun login() {
        var i: Int
        do {
            // Afisăm opțiunile disponibile
            print("1 - Apply discount for the product (in %)\n")
            print("0 - Sign Out\n")
            print("-> ")
            i = KeyboardReader.read()?.toInt() ?: 0

            if (i == 0) {
                break
            }

            if (i == 1) {
                // Notificăm observatorii despre cererea de aplicare a discountului
                viewers.forEach { it.update(this) }
            }
        } while (i != 0)
    }
}
```

```
        }

    } while (i == 1)

}

fun getUsername(): String {
    return this.user
}

fun getPassword(): String {
    return this.password
}

override fun attach(o: Observer) {
    this.viewers.add(o)
}

}

// Clasa MyLogger care implementează interfața Logger
class MyLogger(private val file: String) : Logger {
    override fun writeData(data: String) {
        try {
            // Scriem datele în fișierul de jurnal
            Files.write(Paths.get(file), data.toByteArray(), StandardOpenOption.APPEND)
        } catch (e: IOException) {
            print(e.message + "\n")
            exitProcess(0)
        }
    }
}
```

```
fun main() {  
    // Creăm un utilizator real și îl atașăm ca observator al PriceCalc  
    val radu = RealUser("Raducu", "123456")  
    radu.attach(PriceCalc(Tomatoes(8.00)))  
  
    // Creăm un utilizator proxy și îl autentificăm  
    val user1 = ProxyUser(radu)  
    user1.login()  
}
```

SV81

Kot71 - Utilizând modelul adaptor să se creeze un program Kotlin care va primi la intrare diverse tipuri de date simple sau de tip colecție și va realiza scrierea lor într-un fișier ca o singura operație care primește doar obiectul și numele fișier. Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

P319 - Utilizând modelul comandă să se scrie în Python un program care pornind de la o clasă student va asigura posibilitatea unor comenzi specifice unui obiect colegă care să conducă la schimbarea stării interne a unui obiect student (de ex din fericit în disperat). Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

SV81**Kot71**

Utilizând modelul adaptor să se creeze un program Kotlin care va primi la intrare diverse tipuri de date simple sau de tip colecție și va realiza scrierea lor într-un fișier ca o singura operație care primește doar obiectul și numele fișier. Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

P319

Utilizand modelul comanda sa se scrie in Python un program care pornind de la o clasa student va asigura posibilitatea unor comenzi specifice unui obiect colega care sa conduca la schimbarea starii interne a unui obiectului student (de ex din fericit in disperat). Se vor desena diagrama de clase si de obiecte. Se va explica maniera aplicare a principiilor SOLID.s

PROBLEMA 1 KOTLIN

```
import java.io.BufferedWriter
import java.io.File
import java.lang.Exception

// Definim clasa Adapter
class Adapter{
    val mySerializer = Serializer()

    // Metoda care adaptează un obiect la sir de caractere
    fun AdaptToString(obj: Any) : String{
        try {
            // Verificam tipul obiectului si il adaptam in functie de tip
            if (obj is Int) {
                val leInt = obj.toInt()
                return leInt.toString()
            }
            if (obj is String) {
                return obj.toString()
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
}
```

```

    }

    if (obj is Collection<*>) {
        var s: String = "{ "
        obj.forEach {
            s += it.toString() + " "
        }
        s += "}"
        return s
    }

    return obj.toString()
}

catch (e : Exception){
    return ""
}

}

// Metoda care scrie un obiect într-un fișier folosind serializerul
fun writeToFile(filename: String, obj:Any){
    mySerializer.SerializeToFile(filename, AdaptToString(obj))
}

}

// Definim clasa Serializer
class Serializer{

    // Metoda care serializează un conținut într-un fișier
    fun SerializeToFile(filename: String, content: String){
        try {
            val myFile = File(filename)
            myFile.writeText(content)
        }
    }
}

```

```

        }

    catch(e : Exception)

    {

        println("Eroare la scrierea în fișier!")

    }

}

}

// Definim clasa MyObject

class MyObject(val greet:String, val name:String){

    override fun toString(): String {

        return greet + " " + name + "!"

    }

}

fun main(args: Array<String>) {

    val myAdapter = Adapter()

    val myInt = 42

    val myString = "42 ca sir de caractere"

    val myCollection = listOf(42, "42 ca sir de caractere în colecție", listOf("element în elementul colecției", 32))

    val myObject = MyObject("Salut", "Vlad")



    // Scriem obiectele în fișiere folosind adapterul

    myAdapter.writeToFile("intFile.txt", myAdapter.AdaptToString(myInt))

    myAdapter.writeToFile("stringFile.txt", myAdapter.AdaptToString(myString))

    myAdapter.writeToFile("collectionFile.txt", myAdapter.AdaptToString(myCollection))

    myAdapter.writeToFile("objectFile.txt", myAdapter.AdaptToString(myObject))
}

```

}

SV76

Kot106 - Utilizând Kotlin și OOP să se creeze un program care permite modelarea unei săli de laborator cu tot cu operații pentru obiecte. Se vor folosi liste mutable. Se va desena diagrama UML (clase și obiecte). Se va explica maniera de aplicare a principiilor SOLID.

Py314 - Utilizând modelul pod pot să se scrie un program Python care va desena în mod grafic un triunghi un cerc și un dreptunghi. Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

SV76

Kot106

Utilizand Kotlin si OOP sa se creeze un program care permite modelarea un ei sali de laborator cu tot cu operatii pentru obiecte.Se vor folosi liste mutable.Se vor desena diagrama UML(clase si obiecte).Se va explica maniera de aplicare a principiilor SOLID.

Py314

Utilizand modelul pod pot sa se scrie un program Python care va desena in mod grafic un triunghi un cerc si un dreptunghi.Se vor desena diagrama de clase si de obiecte.Se va explica maniera de aplicare a principiilor SOLID.

PROBLEMA 1 KOTLIN

```
import java.lang.Exception
import kotlin.reflect.jvm.internal.impl.types.AbstractTypeCheckerContext

// Interfața pentru Sala
interface ISala{
    fun setRecuzite(recuzite : List<IRecuzita>, timp : Int) // Setează recuzitele în sala la un anumit timp
    fun intraPersoana(persoana : IPersoana, timp : Int) // O persoană intră în sală la un anumit timp
    fun iesePersoana(persoana : IPersoana, timp : Int) // O persoană iese din sală la un anumit timp
    fun adaugaRecuzita(recuzita : IRecuzita, timp : Int) // Adaugă o recuzită în sală la un anumit timp
    fun stergeRecuzita(recuzita : IRecuzita, timp : Int) // Scoate o recuzită din sală la un anumit timp
    fun persoaneInSala() // Afisează persoanele din sală
    fun recuziteInSala() // Afisează recuzitele din sală
    fun logSala() // Afisează log-ul pentru sala respectivă
}

// Interfața pentru Recuzită
interface IRecuzita{
    fun folositaDe(persoana : IPersoana?) // Setează persoana care folosește recuzita sau null dacă nu este folosită
}

// Interfața pentru Persoană
interface IPersoana{}
```

```

fun vorbeste(mesaj : String) // Persoana rostește un mesaj

fun intraSala(sala : ISala, timp : Int) // Persoana intră într-o sală la un anumit timp

fun ieseSala(sala : ISala, timp : Int) // Persoana iese dintr-o sală la un anumit timp

fun folosesteRecuzita(recuzita : IRecuzita, timp : Int) // Persoana folosește o recuzită la un anumit timp

fun opresteFolosireaRecuzitei(recuzita: IRecuzita, timp: Int) // Persoana oprește folosirea unei recuzite la un anumit timp

}

// Clasa SalaDeLaborator implementează interfața ISala

class SalaDeLaborator(private val numeSala : String) : ISala{

    val recuzite = mutableListOf<IRecuzita>() // Lista recuzitelor din sala

    val persoane = mutableListOf<IPersoana>() // Lista persoanelor din sala

    var logText = "" // Textul log-ului pentru sala

    override fun setRecuzite(recuzite: List<IRecuzita>, timp: Int) {

        recuzite.forEach {

            adaugaRecuzita(it, timp) // Adaugă fiecare recuzită în sala curentă la timpul specificat
        }
    }

    override fun intraPersoana(persoana: IPersoana, timp: Int) {

        persoane.add(persoana) // Adaugă persoana în sala curentă

        log
    }
}

```

PROBLEMA 2 PYTHON

```
import abc

# Interfața pentru forme
class IShape(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def DrawShape(self):
        pass

# Interfața pentru culori
class IColor(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def GetColor(self) -> str:
        pass

# Clasa cerc implementează interfața IShape
class Circle(IShape):
    def __init__(self, radius, color):
        self.radius = radius
        self.color = color

    def DrawShape(self):
        print("Draw circle or radius " + str(self.radius) + " and color " + self.color.GetColor())

# Clasa triunghi implementează interfața IShape
class Triangle(IShape):
    def __init__(self, length1, length2, length3, color):
        self.length1 = length1
```

```

        self.length2 = length2
        self.length3 = length3
        self.color = color

def DrawShape(self):
    lengths = [self.length1, self.length2, self.length3]
    print("Draw triangle or lengths " + str(lengths) + " and color " + self.color.GetColor())

# Clasa dreptunghi implementează interfața IShape
class Rectangle(IShape):
    def __init__(self, width, height, color):
        self.width = width
        self.height = height
        self.color = color

    def DrawShape(self):
        print("Draw rectangle or width " + str(self.width) + " and height " + str(self.height) + " and color " +
              self.color.GetColor())

# Clasa Red implementează interfața IColor
class Red(IColor):
    def GetColor(self) -> str:
        return "Red"

# Clasa Blue implementează interfața IColor
class Blue(IColor):
    def GetColor(self) -> str:
        return "Blue"

```

```
if __name__ == '__main__':
    # Crearea obiectelor și apelarea metodei DrawShape pentru fiecare
    circle = Circle(3, Red())
    triangle = Triangle(4, 5, 6, Blue())
    rectangle = Rectangle(5, 6, Red())

    circle.DrawShape()
    triangle.DrawShape()
    rectangle.DrawShape()
```

SV58

Kot86 - Utilizând Kotlin și OOP să se creeze (KBD sau din program) un grup de oameni care pot efectua (cu totii) următoarele activități: mănâncă, beau, dansează. Fiecare om este diferit deci funcțiile vor primi parametri diferiți. Pornind de la următoarele declaratii să se creeze un program care răspunde la întrebări: ion mănâncă mere, bea bere și dansează cu femei. Vasile mănâncă pere bea vodcă și dansează cu bărbați. Alex bea vin mânâncă prăjitură brune și dansează cu șefii. Exemplu de întrebare: Cu cine poate dansa Vasile și ce poate manca bărbații dar ce poate manca femeile? Se va folosi polimorfismul. Se va desena diagrama UML (clase și obiecte). Se va explica maniera de aplicare a principiilor SOLID.

Py371 - Să se scrie un program Python (utilizând threading) care va utiliza subclase active (cu un fir) pentru procesarea simultană a unui hasmap bazată pe funcții pure. Clasa de bază va stabili operațiile (de ex adunare, scădere înmulțire binară pe 16 biți) iar sub clasele vor pune la dispoziție obiecte care se execută în fire separate..Se va prezenta diagrama de clase și de obiecte..Se vor utiliza semafoare

SV58

Kot86

Utilizând Kotlin și OOP să se creeze (KBD sau din program) un grup de oameni care pot efectua (cu totii) următoarele activități: mananca, beau, danseaza. Fiecare om este diferit deci functiile vor primi parametri diferiti. Pornind de la următoarele declaratii să se creeze un program care raspunde la intrebari: ion mananca mere, bea bere si danseaza cu femei. Vasile mananca pere bea vodca si danseaza cu barbati. Alex bea vin mananca prajituri brune si danseaza cu sefii. Exemple de intrebare: Cu cine poate dansa Vasile si ce poate manca barbatii dar ce poate manca femeile? Se va folosi polimorfism. Se va desena diagrama UML. Se va explica maniera de a aplicare a principiilor SOLID.

Py371

Sa se scrie un program Python care va utiliza subclase active pentru procesarea simultana a unui hashmap bazata pe functii pure. Clase de baza va stabili operatiile iar subclasele vor pune la dispozitie obiecte care se executa in fire separate. Se va prezenta diagrama de clase si de obiecte. Se vor utiliza semafoare.

PROBLEMA 1 KOTLIN

```
class Person(private val name: String, private val gender: String) {  
    private var food: String = ""  
    private var drink: String = ""  
    private var dancingPartner: String = ""  
  
    // Metoda pentru a manca  
    fun eat(food: String) {  
        this.food = food  
    }  
  
    // Metoda pentru a bea  
    fun drink(drink: String) {  
        this.drink = drink  
    }  
  
    // Metoda pentru a dansa  
    fun dance(partner: String) {  
        this.dancingPartner = partner  
    }  
}
```

```
// Metoda pentru a afisa informatii despre persoana
fun whatIdo() {
    println("Eu sunt " + this.name + " - mananc " + this.food + ", beau " + this.drink + " si dansez cu " +
this.dancingPartner)
}

// Metoda pentru a obtine numele persoanei
fun getName(): String {
    return this.name
}

// Metoda pentru a obtine partenerul de dans al persoanei
fun getDancingPartner(): String {
    return this.dancingPartner
}

// Metoda pentru a obtine genul persoanei
fun getGender(): String {
    return this.gender
}

// Metoda pentru a obtine mancarea persoanei
fun getFood(): String {
    return this.food
}

class PersonBuilder() {
    companion object {
```

```
// Metoda pentru a crea o persoana pe baza unui string de intrare
fun getPerson(statement: String, gender: String): Person {
    val slicedStatement = statement.replace(" ", "").split(" ")
    val ret = Person(slicedStatement[0], gender)
    ret.eat(slicedStatement[slicedStatement.indexOf("mananca") + 1])
    ret.drink(slicedStatement[slicedStatement.indexOf("bea") + 1])
    ret.dance(slicedStatement[slicedStatement.indexOf("danseaza") + 2])
    return ret
}
```

```
class GangOfPeople() {
    private val people = mutableListOf<Person>()

    // Metoda pentru a adauga o persoana in lista
    fun addPerson(p: Person) {
        people.add(p)
    }

    // Metoda pentru a afisa informatii despre toate persoanele
    fun whatWeDo() {
        people.forEach { it.whatIdo() }
        println()
    }
}
```

```
// Metoda pentru a obtine partenerul de dans al unei persoane dupa nume
```

```
fun whoDanceWith(name: String?): String {  
    people.forEach {  
        if (it.getName() == name)  
            return it.getDancingPartner()  
    }  
  
    return ""  
}  
  
// Metoda pentru a afisa ce mananca femeile  
fun whatGirlsEat() {  
    people.forEach {  
        if (it.getGender() == "femeie")  
            println(it.getName() + " mananca " + it.getFood())  
    }  
}  
  
// Metoda pentru a afisa ce mananca barbatii  
fun whatBoysEat() {  
    people.forEach {  
        if (it.getGender() == "barbat")  
            println(it.getName() + " mananca " + it.getFood())  
    }  
}  
  
class GangAnswerer(private val gang: GangOfPeople) {  
    // Metoda pentru a raspunde la intrebare  
    fun answer(question: String) {
```

```

val questions = mutableListOf<String>()

val aux = question.replace("?", "").split(" si ")

aux.forEach { it.split("dar").forEach { questions.add(it) } }

questions.forEach {
    println(it + "?")

    if (it.contains("u cine poate dansa")) {
        val name = it.split(" ").lastOrNull()

        println(name + " poate dansa cu " + gang.whoDanceWith(name))
    }
}

if (it.contains("ce pot manca")) {
    val gender = it.split(" ").lastOrNull()

    if (gender == "femeile")
        gang.whatGirlsEat()
    else
        gang.whatBoysEat()
}
// de completat...
println()
}

}

fun main() {
    // Creăm o instanță a clasei GangOfPeople
    val gang = GangOfPeople()
}

```

```

// Creăm persoane și le adăugăm în gang
gang.addPerson(PersonBuilder.getPerson("Ion mananca mere, bea bere si danseaza cu femei",
"barbat"))

gang.addPerson(PersonBuilder.getPerson("Vasile mananca pere, bea vodca si danseaza cu barbati",
"barbat"))

gang.addPerson(PersonBuilder.getPerson("Alex bea vin, mananca prajitui si danseaza cu sefii",
"barbat"))

gang.addPerson(PersonBuilder.getPerson("Lavinia bea suc, mananca tort si danseaza cu Vlad",
"femeie"))

// Afisăm informațiile despre persoanele din gang
gang.whatWeDo()

// Creăm un obiect GangAnswerer și răspundem la întrebarea dată
val q_and_a = GangAnswerer(gang)

q_and_a.answer("Cu cine poate dansa Vasile și ce pot manca barbatii dar ce pot manca femeile?")

}

```

PROBLEMA 2 PYTHON

```

from threading import Thread
from threading import Lock
import time

class Command(Thread):

```

```

def __init__(self, hm):
    super().__init__()
    self._hm = hm
    self._sem = Lock() # Creăm un obiect de tip Lock pentru a sincroniza accesul la resurse comune
    self._is_running = False

def execute(self):
    if not self._is_running:
        self.start() # Începe executarea într-un thread separat
        self._is_running = True
    else:
        self.run() # Execută metoda run în thread-ul principal

# Metoda run trebuie implementată în clasele derivate
def run(self) -> None:
    raise NotImplementedError("Subclasses must implement run()")

class HashMapProcess:
    def __init__(self, h):
        self.__adder = HashMapAdd(h)
        self.__sub = HashMapSub(h)
        self.__mul = HashMapMul(h)

    def add(self):
        self.__adder.execute()

    def sub(self):
        self.__sub.execute()

```

```

def mul(self):
    self.__mul.execute()

class HashMapAdd(Command):
    def __init__(self, h):
        super().__init__(h)

    def run(self) -> None:
        self._sem.acquire() # Acquire the lock, blocking if necessary
        s = 0
        for i in self._hm:
            if self._hm[i] <= pow(2, 16) - 1:
                s += self._hm[i]
            else:
                raise Exception("Error! Number is too big!")
        print("Sum: " + str(s))
        print("Now sleeping...")
        print()
        time.sleep(1) # Simulăm o întârziere de 1 secundă
        self._sem.release() # Eliberăm lock-ul

```

```
class HashMapSub(Command):
```

```

    def __init__(self, h):
        super().__init__(h)
```

```
    def run(self) -> None:
```

```
self._sem.acquire() # Acquire the lock, blocking if necessary
s = 0
for i in self._hm:
    if self._hm[i] <= pow(2, 16) - 1:
        s -= self._hm[i]
    else:
        raise Exception("Error! Number is too big!")
print("Sub: " + str(s))
print("Now sleeping...")
print()
time.sleep(1) # Simulăm o întârziere de 1 secundă
self._sem.release() # Eliberăm lock-ul
```

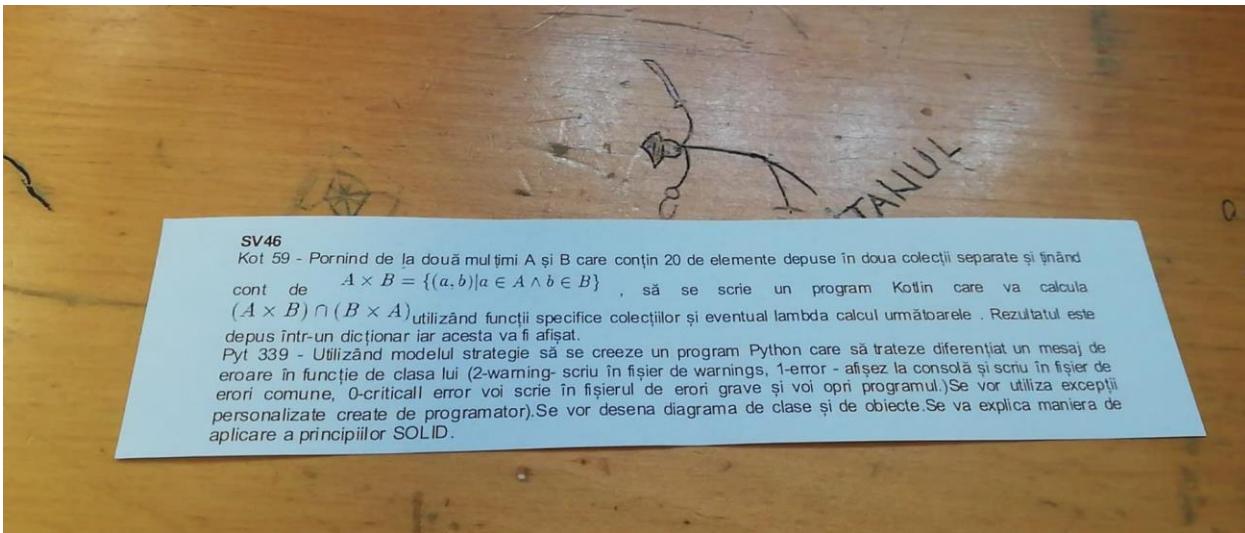
```
class HashMapMul(Command):
    def __init__(self, h):
        super().__init__(h)

    def run(self) -> None:
        self._sem.acquire() # Acquire the lock, blocking if necessary
        p = 1
        for i in self._hm:
            if self._hm[i] <= pow(2, 16) - 1:
                p *= self._hm[i]
            else:
                raise Exception("Error! Number is too big!")
        print("Mul: " + str(p))
        print("Now sleeping...")
        print()
```

```
time.sleep(1) # Simulăm o întârziere de 1 secundă
self._sem.release() # Eliberăm lock-ul

if __name__ == "__main__":
    my_dict = {0: 10, 1: 20, 2: 30, 3: 40, 4: 50}
    hp = HashMapProcess(my_dict)

    hp.add() # Execută operația de adunare într-un thread separat
    hp.add() # Execută operația de adunare într-un thread separat
    hp.mul() # Execută operația de înmulțire într-un thread separat
    hp.sub() # Execută operația de scădere într-un thread separat
    hp.add() # Execută operația de adunare într-un thread separat
    hp.mul() # Execută operația de înmulțire într-un thread separat
    hp.add() # Execută operația de adunare într-un thread separat
    hp.add() # Execută operația de adunare într-un thread separat
    hp.sub() # Execută operația de scădere într-un thread separat
```



SV46

Kot 59 - Pornind de la două mulțimi A și B care conțin 20 de elemente depuse în două colecții separate șiținând cont de $A \times B = \{(a, b) | a \in A \wedge b \in B\}$, să se scrie un program Kotlin care va calcula $(A \times B) \cap (B \times A)$ utilizând funcții specifice colecțiilor și eventual lambda calcul următoarele. Rezultatul este depus într-un dicționar iar acesta va fi afișat.

PYT 339 - Utilizând modelul strategie să se creeze un program Python care să trateze diferențiat un mesaj de eroare în funcție de clasa lui (2-warning scriu în fișier de warnings, 1-error - afișez la consolă și scriu în fișier de erori comune, 0-critical error vor scrie în fișierul de erori grave și voi opri programul.) Se vor utiliza exceptii personalizate create de programator. Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

Kot 59

Pornind de la două multimi A si B care contin 20 de elemente depuse4 in doua colectii separate si tinand cont de AXB={}, sa se scrie un program Kotlin care va calcula utilizant functii specifice colectiilor si eventual loambda calclul urmatoarele .Rezultatul este depus intr un dictionar iar acesta va fi afisat.

PYT 339

Utilizand modelul strategie sa se creeze un program Python care sa trateze diferențiat un mesaj de eroare in functie de clasa lui (2 warning scriu in fisier de warnings, 1 error afisez la consola si scriu in fisier de eroari comune,0-critical error vor scrie in fisierul de erori grave si voi opri programul).Se vor utiliza exceptii personalizate4 create de programator).Se

vor desena diagrama de clase si de obiecte.Se vaq explica maniera de aplicare a principiilor SOLID.

PROBLEMA 2 PYTHON

```
import abc
```

```
from abc import ABC

class MyException(Exception):

    def __init__(self, message, type):
        super(MyException, self).__init__(message)
        self.message = message
        self.type = type

    def getType(self):
        return self.type

    def getMessage(self):
        return self.message

class IStrategy(metaclass=abc.ABCMeta):

    @abc.abstractmethod
    def treat(self, message):
        pass

class ErrorStrategy(IStrategy):

    def treat(self, message):
        f = open("common_errors.txt", "a") # Deschidem fișierul "common_errors.txt" în modul adăugare ("a")
        f.write("Error: " + message) # Scriem mesajul de eroare în fișier
        print("Error: " + message) # Afişăm mesajul de eroare pe consolă
```

```
class WarningStrategy(IStrategy):
    def treat(self, message):
        f = open("warnings.txt", "a") # Deschidem fișierul "warnings.txt" în modul adăugare ("a")
        f.write("Warning: " + message) # Scriem mesajul de avertizare în fișier

class CriticalStrategy(IStrategy):
    def treat(self, message):
        f = open("critical_errors.txt", "a") # Deschidem fișierul "critical_errors.txt" în modul adăugare ("a")
        f.write("Critical: " + message) # Scriem mesajul de eroare critică în fișier
        exit(-1) # Ieșim din program cu codul de eroare -1

class UnknownStrategy(IStrategy):
    def treat(self, message):
        print("Unknown: " + message) # Afisăm mesajul de eroare necunoscut pe consolă

class IExceptionClassManager(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def createTreatStrategy(self, customException) -> IStrategy:
        pass

    @abc.abstractmethod
    def resolveException(self):
        pass
```

```

class MyExceptionManager(IExceptionClassManager):

    def __init__(self):
        self.myStrategy = UnknownStrategy() # Strategia implicită este UnknownStrategy
        self.message = ""

    def createTreatStrategy(self, customException):
        if isinstance(customException, MyException):
            self.message = customException.getMessage() # Obținem mesajul din excepția personalizată
            typeNumber = customException.getType() # Obținem tipul din excepția personalizată
            if typeNumber == 2:
                self.myStrategy = WarningStrategy() # Setăm strategia de avertizare pentru tipul 2
                return
            if typeNumber == 1:
                self.myStrategy = ErrorStrategy() # Setăm strategia de eroare pentru tipul 1
                return
            if typeNumber == 0:
                self.myStrategy = CriticalStrategy() # Setăm strategia de eroare critică pentru tipul 0
                return
            self.myStrategy = UnknownStrategy() # Setăm strategia necunoscută pentru alte tipuri de excepții
        else:
            self.myStrategy = UnknownStrategy() # Setăm strategia necunoscută pentru alte tipuri de excepții

    def resolveException(self):
        self.myStrategy.treat(self.message) # Tratăm excepția folosind strategia corespunzătoare

if __name__ == '__main__':
    case = 0
    exceptionManager = MyExceptionManager()

```

```
try:  
    if case == 2:  
        raise MyException("MyWarning Message", 2) # Ridicăm o excepție de tip avertizare  
  
    if case == 1:  
        raise MyException("MyError Message", 1) # Ridicăm o excepție de tip eroare  
  
    if case == 0:  
        raise MyException("MyCritical Message", 0) # Ridicăm o excepție de tip eroare critică  
except MyException as e:  
    exceptionManager.createTreatStrategy(e) # Creăm strategia de tratare a excepției  
    exceptionManager.resolveException() # Tratăm excepția conform strategiei corespunzătoare
```

FISIERE

common_errors.txt

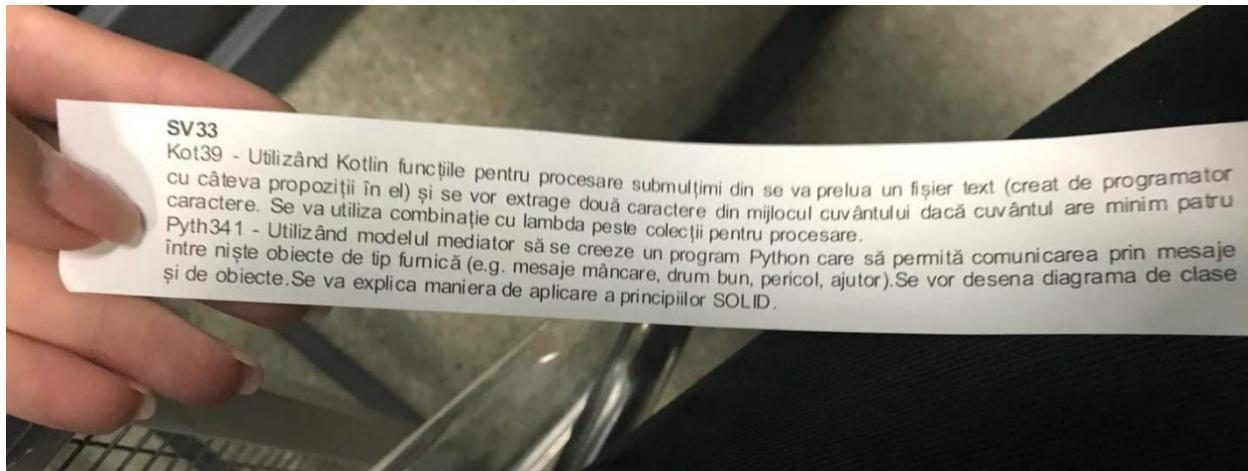
Error: MyError Message

critical_errors.txt

Critical: MyCritical Message

warnings.txt

Warning: MyWarning Message



SV33

Kot39

Utilzand Kotlin functiile pentru procesare submultimi din se va prelua un fisier text(creat de programator cu cateva propozitii in el) si se vor extrage doua caractere din mijlocul cuvantul daca cuvanatul are minim patru caractere.Se va utiliza combinatie cu lambda peste colectii pentru procesare.

Pyth341

Utilizand modelul mediator sa se creeze un program Python care sa permita comunicarea prin mesaje intre niste obiecte de tip furnica.Se vor desena diagrama de 4 clase si de obiecte .Se va explica maniera de aplicare a principiilor SOLID.

PROBLEMA 1 KOTLIN

```
import java.io.File  
import java.lang.Exception
```

```

fun main() {
    try {
        File("text").forEachLine { // Iterăm prin fiecare linie a fișierului "text"
            it.split(" ").filter { it.length >= 4 } // Splituim linia în cuvinte și filtrăm doar cuvintele cu lungimea
            >= 4
                .map { word -> Pair<Char, Char>(word[word.length / 2 - 1], word[word.length / 2]) } // Pentru
                fiecare cuvânt, creăm un Pair format din caracterele de la mijlocul cuvântului
                .forEach { println(it) } // Afisăm fiecare Pair pe o linie separată
        }
    } catch (e: Exception) {
        print("Error: " + e.printStackTrace()) // Afisăm mesajul de eroare și stack trace-ul
    }
}

```

PROBLEMA 2 PYTHON

```

from abc import ABC

class Mediator(ABC):
    def notify(self, sender, event: str, receiver):
        raise NotImplementedError("Not implemented!")

```

```

class AntsMessageMediator(Mediator):
    def __init__(self, ants_squad: list):

```

```
self.__squad = ants_squad

for ant in self.__squad:
    ant.set_mediator(self)

def notify(self, sender, event: str, receiver):
    # Verifică tipul de eveniment și trimite mesajul corespunzător anturajului

    if event == "farewell":
        for ant in self.__squad:
            if ant == receiver:
                ant.farewell_react(sender)
                return

    if event == "food":
        for ant in self.__squad:
            if ant == receiver:
                ant.answer_to_food_request(sender)
                return

    if event == "warning":
        for ant in self.__squad:
            if ant == receiver:
                ant.run(sender)
                return

    if event == "help":
        for ant in self.__squad:
            if ant == receiver:
                ant.help_someone(sender)
                return
```

```
raise Exception("Unknown message!")

class Ant:

    def __init__(self, name: str, mediator: Mediator = None):
        self._mediator = mediator
        self._name = name

    def get_name(self) -> str:
        return self._name

    def set_mediator(self, mediator: Mediator):
        self._mediator = mediator

class RedAnt(Ant):

    def __init__(self, name: str):
        super().__init__(name)

    def farewell_react(self, sender: Ant):
        print("Thank you, " + str(sender.get_name()) + "! - from " + str(self._name) + " the red ant\n")

    def say_farewell(self, to: Ant):
        # Anta roşie îşi ia rămas bun de la altă furnică
        print("I'm " + str(self._name) + " the red ant and I say farewell to " + str(to.get_name()))
        self._mediator.notify(self, "farewell", to)

    def ask_for_food(self, to: Ant):
```

```
# Anta roşie cere hrană de la altă furnică
print("I'm " + str(self._name) + " the red ant and I want some food from " + str(to.get_name()))
self._mediator.notify(self, "food", to)

def answer_to_food_request(self, sender: Ant):
    # Anta roşie răspunde cererii de hrană a aliei furnici
    print("There you have some bread crumbs, " + str(sender.get_name()) + "! - from " + str(
        self._name) + " the red ant\n")

def warn_someone(self, to: Ant):
    # Anta roşie avertizează o altă furnică
    print("I'm " + str(self._name) + " the red ant. Look, " + str(to.get_name()) + ", we are in danger!")
    self._mediator.notify(self, "warning", to)

def run(self, sender: Ant):
    # Anta roşie începe să alerge în urma unei instrucțiuni de la o altă furnică
    print("Now I'm running, " + str(sender.get_name()) + "! - from " + str(self._name) + " the red ant\n")

def ask_for_help(self, to: Ant):
    # Anta roşie cere ajutor de la o altă furnică
    print("I'm " + str(self._name) + " the red ant and I need some help from " + str(to.get_name()))
    self._mediator.notify(self, "help", to)

def help_someone(self, sender: Ant):
    # Anta roşie vine în ajutorul aliei furnici
    print("I will help you, " + str(sender.get_name()) + "! - from " + str(self._name) + " the red ant\n")

if __name__ == '__main__':
```

```
list_of_ants = [RedAnt("Jeff"), RedAnt("Alex"), RedAnt("Lulu"), RedAnt("Maria")]

a = AntsMessageMediator(list_of_ants)

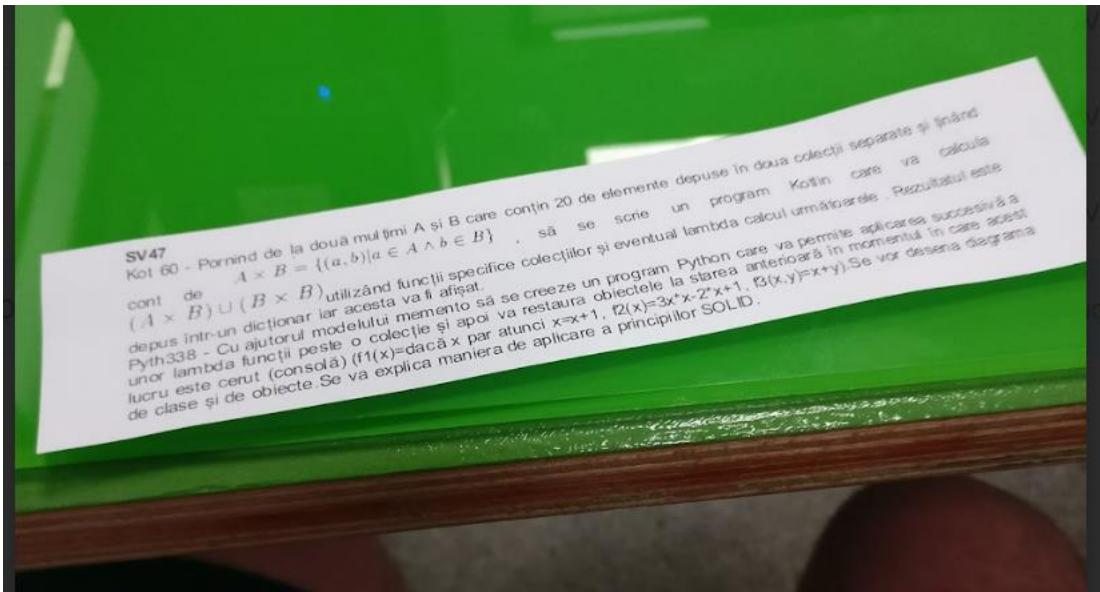
# Exemple de interacțiuni între furnici

list_of_ants[0].say_farewell(list_of_ants[1]) # Anta roșie îi spune rămas bun altei furnici

list_of_ants[1].ask_for_food(list_of_ants[0]) # Anta roșie cere hrănă de la alta furnică

list_of_ants[3].warn_someone(list_of_ants[2]) # Anta roșie avertizează o alta furnică

list_of_ants[2].ask_for_help(list_of_ants[0]) # Anta roșie cere ajutor de la alta furnică
```



SV47

Kot 60

Pornind de la două multimi A și B care conțin 20 de elemente depuse în două colecții separate și tinând cont de $A \times B = \{(a, b) | a \in A \wedge b \in B\}$, să se scrie un program Kotlin care va calcula $(A \times B) \cup (B \times A)$ utilizând funcții specifice colecțiilor și eventual lambda calcul următoarele. Rezultatul este depus într-un dicționar iar acesta va fi afișat.

Pyth338

Cu ajutorul modelului memmmento să se creeze un program Python care va permite aplicarea succesivă a unor lambda funcții peste o colectie și apoi va restaura obiecte la starea anterioara în momentul în care acest lucru este cerut ($f1(x)=$ dacă x par atunci $x=x+1$, $f2(x)=3x^2-x+1$, $f3(x,y)=x+y$). Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

PROBLEMA 1 KOTLIN

```

fun main() {
    val A = listOf(1, 2, 3) // Definirea listei A cu elementele 1, 2, 3
    val B = listOf(3, 4, 5) // Definirea listei B cu elementele 3, 4, 5

    val AxB = A.flatMap { a -> B.map { b -> a to b } } // Produsul cartezian între A și B
    val BxB = B.flatMap { a -> B.map { b -> a to b } } // Produsul cartezian între B și B

    println("AxB: " + AxB) // Afisarea produsului cartezian AxB
    println("BxB: " + BxB) // Afisarea produsului cartezian BxB

    val AxB_U_BxB = (AxB + BxB).distinct() // Reuniunea dintre AxB și BxB, eliminând duplicitările
    print("AxB U BxB: " + AxB_U_BxB) // Afisarea reuniunii AxB_U_BxB
}

```

PROBLEMA 2 PYTHON

```
from abc import ABC, abstractmethod
```

```

class Memento(ABC):
    @abstractmethod
    def get_state(self):
        pass

```

```

class Originator:
    _state = None

```

```

def __init__(self, state):
    self._state = state

def f1(self):
    # Aplică o funcție de transformare asupra stării, adăugând 1 la elementele pare
    self._state = list(map(lambda x: x + 1 if x % 2 == 0 else x, self._state))

def f2(self):
    # Aplică o funcție de transformare asupra stării, calculând o expresie polinomială pentru fiecare
    # element
    self._state = list(map(lambda x: (3 * x * x) - (2 * x) + 1, self._state))

def f3(self):
    # Aplică o funcție de transformare asupra stării, adunând fiecare element cu el însuși
    self._state = list(map(lambda x, y: x + y, self._state, self._state))

def save(self) -> Memento:
    # Creează un obiect Memento care conține starea curentă a Originator-ului
    return ConcreteMemento(self._state)

def restore(self, memento: Memento):
    # Restaurează starea Originator-ului la cea din Memento
    self._state = memento.get_state()

def get_list(self) -> list:
    # Returnează starea curentă a Originator-ului
    return self._state

```

```
class ConcreteMemento(Memento, ABC):

    def __init__(self, state):
        self._state = state

    def get_state(self) -> list:
        # Returnează starea conținută în Memento
        return self._state


class Caretaker:

    def __init__(self, originator: Originator):
        self._mementos = [] # Lista de obiecte Memento
        self._originator = originator

    def backup(self):
        print("Saving state...")
        # Salvează starea curentă a Originator-ului într-un obiect Memento
        self._mementos.append(self._originator.save())

    def undo(self):
        if not len(self._mementos):
            return

        # Extrage ultimul Memento din lista
        memento = self._mementos.pop()
        print("Restoring state...")
```

```
try:  
    # Restaurează starea Originator-ului la cea din Memento  
    self._originator.restore(memento)  
  
except Exception:  
    # Dacă nu se poate restaura starea, apelează recursiv undo()  
    self.undo()  
  
  
  
  
if __name__ == '__main__':  
    my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
    o = Originator(my_list)  
    caretaker = Caretaker(o)  
  
  
    caretaker.backup() # Salvează starea inițială  
    print(o.get_list()) # [1, 2, 3, 4, 5, 6, 7, 8, 9]  
  
  
    o.f1() # Aplică transformarea f1  
    print(o.get_list()) # [1, 3, 3, 5, 5, 7, 7, 9, 9]  
  
  
    caretaker.undo() # Restaurează starea inițială  
    print(o.get_list()) # [1, 2, 3, 4, 5, 6, 7, 8, 9]  
  
  
    caretaker.backup() # Salvează starea curentă  
    o.f2() # Aplică transformarea f2  
    print(o.get_list()) # [10, 17, 28, 43, 62, 85, 112, 143, 178]  
  
  
    caretaker.undo() # Restaurează starea anterioară  
    print(o.get_list()) # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
caretaker.backup() # Salvează starea curentă  
o.f3() # Aplică transformarea f3  
print(o.get_list()) # [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
caretaker.undo() # Restaurează starea anterioară  
print(o.get_list()) # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```



Simulare de trimitere examen PP/...



examenppsd@gmail.com

Me



Today, 11:18 AM

ilizând genericile să se implementeze în Kotlin o funcție extensie pe un subarbore de tipuri de date (vezi ierarhia de tipuri de date cu limitare superioară din curs/laborator) care să verifice dacă variabila conține un număr și apoi să determine dacă numărul este prim sau nu.

utilizand genericile sa se implementeze in Kotlin o functie extensie pe un subarbore de tipuri de date(vezi ierarhia de tipuri de date cu limitare superioara din curs/laborator)care sa verifice daca variabila contine un numar si apoi sa se determine daca numarul este primi sau nu.

```
fun List<Number>.allPrimes(): Boolean {  
    // Verifică dacă toate elementele din listă sunt numere prime
```

```

        return !this.any { it !is Int || !it.isPrime() }

    }

fun Int.isPrime(): Boolean {
    // Verifică dacă numărul este prim
    if (this < 0)
        return false
    if (this in 1..2)
        return true
    // Verifică dacă există vreun divizor între 2 și numărul însuși
    return !2.until(this / 2 + 1).any { this % it == 0 }
}

fun main() {
    val lists = listOf(
        listOf(1, 2, 3, 4, 5),
        listOf(1.0, 2.0, 3.0),
        listOf(1, 3, 7, 11)
    )
    // Iterează prin fiecare listă și afișează rezultatul verificării de numere prime
    lists.forEachIndexed { index, list -
        println("List $index: $list -> all are primes: ${list.allPrimes()}")
    }
}

```

}

se implementeze în Python un automat cu trei stări în care să se proceseze cu expresii lambda o listă de numere întregi trimisă ca argument în constructor astfel:

În s0 se verifică dacă mai sunt elemente în listă (dacă da, se duce în starea s1 unde se identifică și se șterge primul element prim găsit apoi se trece în s2).

În s2 se identifică și se șterge primul element neprim găsit apoi se trece în s0.

La fiecare ștergere se afișează elementul sters.

În s0 dacă nu mai există elemente, se anunță acest lucru, iar programul se termină.

Sa se implementeze in Python un automat cu trei stari in care sa se proceseze cu expresii lambda o lista de numere intregi trimisa ca argument in constructor astfel:

in s0 se verifica daca mai sunt elemente in lista(daca da,se duce in starea s1 unde se identifice si se sterge primul element prim gasit apoi se trece in s2).

In s2 se identifica si se sterge primul element neprim gasit apoi se trece in s0.La fiecare stergere se afiseaza elementul sters.In s0 daca nu mai exista elemente,se anunta acest lucru,iar programul se termina.

FSM.py

```
from abc import ABCMeta, abstractmethod
```

```
from functional import seq
```

```
def isPrime(x):
```

```
    # Verifică dacă un număr este prim
```

```
    if x < 0:
```

```
        return False
```

```
    if x == 1 or x == 2:
```

```

        return True

# Verifică dacă există vreun divizor între 2 și jumătatea numărului

if seq(range(2, x // 2 + 1)).filter(lambda it: x % it == 0).len() > 0:
    return False

return True


class FSM:

    def __init__(self, l):
        self._l = l

        self._states = [State0(self), State1(self), State2(self)]
        self._current_state = self._states[0]

    def get_list(self):
        return self._l

    def get_state(self, state_number):
        # Obține starea corespunzătoare unui număr de stare dat
        if state_number >= len(self._states) or state_number < 0:
            return None
        return self._states[state_number]

    def set_state(self, state):
        # Setează starea curentă a FSM-ului
        self._current_state = state

    def do_task(self):
        # Execută sarcina stării curente
        return self._current_state.do_task()

```

```
class State(metaclass=ABCMeta):

    def __init__(self, fsm):
        self._fsm = fsm

    @abstractmethod
    def do_task(self):
        pass


class State0(State):

    def __init__(self, fsm):
        super().__init__(fsm)

    def do_task(self):
        # Verifică dacă lista are elemente și trece la starea următoare
        if len(self._fsm.get_list()) > 0:
            self._fsm.set_state(self._fsm.get_state(1))
            return True
        return False


class State1(State):

    def __init__(self, fsm):
        super().__init__(fsm)

    def do_task(self):
        l = self._fsm.get_list()
        # Găsește primul element prim din listă și îl elimină
        found = seq(l).find(lambda it: isPrime(it))
        if found != None:
            print(f"S-a eliminat elementul prim {found}.")
```

```

        l.remove(found)
        self._fsm.set_state(self._fsm.get_state(2))
        return True

class State2(State):
    def __init__(self, fsm):
        super().__init__(fsm)

    def do_task(self):
        l = self._fsm.get_list()
        # Găsește primul element neprim din listă și îl elimină
        found = seq(l).find(lambda it: not isPrime(it))
        if found != None:
            print(f"S-a eliminat elementul neprim {found}.")
            l.remove(found)
        self._fsm.set_state(self._fsm.get_state(0))
        return True

```

MAIN.py

```

import FSM

if __name__ == '__main__':
    l = list(range(10)) # Inițializarea unei liste de numere de la 0 la 9
    fsm = FSM.FSM(l) # Inițializarea unui automat finit cu lista de numere
    while fsm.do_task(): # Execută sarcina automatului finit până când nu mai poate trece la o nouă stare

```

pass

ilizând modelul comandă să se scrie în Kotlin un program care pornind de la o clasă student va asigura posibilitatea unor comenzi specifice unui obiect colegă care să conducă la schimbarea stării interne a unui obiect student (de exemplu din fericit în disperat). Apoi acestea vor fi utilizate într-un automat static (cu două stări fericit-nefericit) implementat utilizând modelul stare. Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

Utilizând modelul comandă să se scrie în Kotlin un program care pornind de la o clasă student va asigura posibilitatea unor comenzi specifice unui obiect colegă care să conduca la schimbarea stării interne a unui obiect student (de exemplu din fericit în disperat). Apoi acestea vor fi utilizate într-un automat static (cu două stări fericit-nefericit) implementat utilizând modelul stare. Se vor desena diagrame de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

STUDENTI.py

```
import State  
from random import choice
```

```

class Student:

    def __init__(self, name):
        self.__name = name # Numele studentului
        self.__current_state = "Fericit" # Starea curentă a studentului
        self.__general_states = [State.FericitState(self), State.NefericitState(self)] # Lista de stări generale
        self.__general_state = self.__general_states[0] # Starea generală curentă
        self.__nice_action = lambda colegа: colegа.act_on("Nice") # Acțiunea "Nice" a studentului asupra unui coleg
        self.__bad_action = lambda colegа: colegа.act_on("Bad") # Acțiunea "Bad" a studentului asupra unui coleg

    def get_current_state(self):
        return self.__current_state # Returnează starea curentă a studentului

    def toggle_general_state(self):
        if self.__general_state == self.__general_states[0]:
            self.__general_state = self.__general_states[1]
        else:
            self.__general_state = self.__general_states[0]

    def act_nice(self, colegа):
        self.__current_state = self.__nice_action(коlega) # Execută acțiunea "Nice" asupra unui coleg
        self.__update_general_state() # Actualizează starea generală

    def act_bad(self, colegа):
        self.__current_state = self.__bad_action(коlega) # Execută acțiunea "Bad" asupra unui coleg
        self.__update_general_state() # Actualizează starea generală

```

```

def __update_general_state(self):
    self.__general_state.update_state() # Actualizează starea generală a studentului

def print_state(self):
    print(
        f"Studentul {self.__name} este {self.__current_state}, în general fiind
{self.__general_state.show_state()}.")

class Colega:
    def __init__(self, name):
        self.__name = name # Numele colegului
        self.__fericit_states = ["Fericit", "Bucuros", "Vesel"] # Stările "fericit"
        self.__nefericit_states = ["Nefericit", "Disperat", "Suparat"] # Stările "nefericit"

    def act_on(self, action_type):
        if action_type == "Nice":
            selected = choice(self.__fericit_states) # Selectează aleatoriu o stare "fericit"
        elif action_type == "Bad":
            selected = choice(self.__nefericit_states) # Selectează aleatoriu o stare "nefericit"
        else:
            raise Exception("Actiune incorecta!")
        print(f"Colega {self.__name} a facut un coleg {selected}.")
        return selected # Returnează starea selectată

```

STATE.py

```
from abc import ABCMeta, abstractmethod

class StudentState(metaclass=ABCMeta):

    def __init__(self, student):
        self._student = student # Referință către studentul asociat stării

    @abstractmethod
    def update_state(self):
        pass

    @abstractmethod
    def show_state(self):
        pass

class FericitState(StudentState):

    def __init__(self, student):
        super().__init__(student)

        # Lista stărilor "ok" pentru starea "fericit"
        self.__ok_states = ["Fericit", "Bucuros", "Vesel"]

    def update_state(self):
        if self._student.get_current_state() not in self.__ok_states:
            self._student.toggle_general_state() # Comută starea generală a studentului
```

```

def show_state(self):
    return "Fericit" # Returnează numele stării

class NefericitState(StudentState):
    def __init__(self, student):
        super().__init__(student)

        # Lista stărilor "ok" pentru starea "nefericit"
        self.__ok_states = ["Nefericit", "Disperat", "Suparat"]

    def update_state(self):
        if self._student.get_current_state() not in self.__ok_states:
            self._student.toggle_general_state() # Comută starea generală a studentului

    def show_state(self):
        return "Nefericit" # Returnează numele stării

```

MAIN.py

```

import Studenti

if __name__ == '__main__':
    s1 = Studenti.Student("Florin")
    c1 = Studenti.Colega("Andreea")
    s1.act_bad(c1)
    s1.print_state()
    s1.act_nice(c1)
    s1.print_state()

```

← Simulare de trimitere examen
PP/SD - Plop Andrei, 1211A



examenppsd@gmail.com

Eu



Azi, 11:18

ilizând modelul pod să se scrie în Python un program care va porni de la un model mamifer și apoi va permite instanțiere de obiecte de tip om, femeie, câine și pisică. Aceste obiecte li se vor adăuga 3-4 funcționalități suplimentare la alegere utilizând decoratorii (de exemplu o funcție care descrie/permite interacțiunea dintre om și femeie sau om și pisica etc). Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

Utilizând modelul pod să se scrie în Python un program care va porni de la un model mamifer și apoi va permite instantiere de obiecte de tip om, femeie, câine și pisică. Aceste obiecte li se vor adăuga 3-4 funcționalități suplimentare la alegere utilizând decoratorii (de exemplu o funcție care descrie/permite interacțiunea dintre om și femeie sau om și pisica etc). Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

MAMIFERE.py

```
from abc import ABCMeta, abstractmethod
```

```
# Definirea clasei abstracte "MamiferAbstraction" cu metodele abstracte "speak" și "preferred_food"

# Această clasă servește ca o interfață pentru clasele "Mamifer" și "MamiferImplementation"

class MamiferAbstraction(metaclass=ABCMeta):

    def __init__(self, implementation):
        self._implementation = implementation

    @abstractmethod
    def speak(self):
        pass

    @abstractmethod
    def preferred_food(self):
        pass

class Mamifer(MamiferAbstraction):

    def __init__(self, implementation):
        super().__init__(implementation)

    def speak(self):
        self._implementation.speak()

    def preferred_food(self):
        self._implementation.preferred_food()

class MamiferImplementation(metaclass=ABCMeta):

    @abstractmethod
```

```
def speak(self):
    pass

@abstractmethod
def preferred_food(self):
    pass

# Clasa "Femeie" care implementează interfața "MamiferImplementation"
# Aceasta definește comportamentul specific pentru o femeie
class Femeie(MamiferImplementation):
    def speak(self):
        print("Vorbeste cu voce de femeie.")

    def preferred_food(self):
        print("Caviar")

# Clasa "Barbat" care implementează interfața "MamiferImplementation"
# Aceasta definește comportamentul specific pentru un bărbat
class Barbat(MamiferImplementation):
    def speak(self):
        print("Vorbeste cu voce de barbat.")

    def preferred_food(self):
        print("Mici")

# Clasa "Caine" care implementează interfața "MamiferImplementation"
# Aceasta definește comportamentul specific pentru un câine
```

```

class Caine(MamiferImplementation):
    def speak(self):
        print("Ham ham.")

    def preferred_food(self):
        print("Carne")

# Clasa "Pisica" care implementează interfața "MamiferImplementation"
# Aceasta definește comportamentul specific pentru o pisică
class Pisica(MamiferImplementation):
    def speak(self):
        print("Miau miau.")

    def preferred_food(self):
        print("Peste")

```

MAIN.py

```

from Mamifere import *

if __name__ == '__main__':
    # Crearea instanțelor pentru diverse mamifere folosind clasele și implementările definite anterior
    caine = Mamifer(Caine())
    pisica = Mamifer(Pisica())
    femeie = Mamifer(Femeie())
    barbat = Mamifer(Barbat())

```

```
# Definirea unei liste cu toate instanțele de mamifere create
```

```
mamifere = [caine, pisica, femeie, barbat]
```

```
# Parcurgerea listei de mamifere și apelarea metodelor "speak" și "preferred_food" pentru fiecare
```

```
for mamifer in mamifere:
```

```
    mamifer.speak()
```

```
    mamifer.preferred_food()
```

se scrie un program Python care utilizând modelul strategie va alege în funcție de reacția unui coleg un banc dintr-un dicționar care are seturi de bancuri bune, foarte bune și cele mai bune (un fișier text care are un banc pe paragraf și la început are una/două/trei steluțe (codificarea calității bancului). Deci este nevoie de o clasă student, o metodă de a spune bancuri și una de a asculta și furniza o reacție.

Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

Se scrie un program Python care utilizand modelul strategie va alege in functie de reactia unui coleg un banc dintr un dictionar care are seturi de bancuri bune,foarte bune si cele mai bune(un fisier text care are un banc pe paragraf si la inceput are una/doua/tre stelute (codificarea calitatii bancului).Deci este nevoie de o clasa student,o metoda de a spune bancuri si una de a asculta si furniza o reactie.Se vor desena diagrama de clase si de obiecte . Se va explica maniera de aplicare a principiilor SOLID.

*Vine Bula la scoala cu o buza umflata. Profesoara il intreaba:

- Ce ai patit, Bula?
- Ei, am fost cu tata la pescuit si mi s-a asezat o viespe pe buza.
- Vaaaai, saracul de tine! Si te-a intepat?
- Nu, n-a apucat, a omorat-o tata cu vasla...

**Bula avea meci si nu stia cum sa scape de la scoala. In timpul orei, incepe sa-si ia colegii la bataie.

Profa, tanara:

-Bula, vezi ca te scot afara!

Bula:

-Asta vreau!

Totusi , profa nu-l scoate.

Se enerveaza Bula , isi scoata p..a si incepe sa si-o falfaie.

-Bula, te scot afara!

-Asta vreau!

Tot nu-l scoate.

Bula era nervos , ca mai erau 5 min. pana incepea meciul.

-Da' ce v-as f?te o data!

Profa, mirata:

-Afara!

-Yessss!!!

-Nu tu, Bula! Restul!

***Micul Bula intra in camera parintilor si o vede pe maica-sa cu fusta ridicata, iar pe taica-sau in spatele ei.

Vazandu-l, tatal rade, ii face semn sa plece si sa inchida usa. Dupa ce termina treaba, tatal se duce sa vada ce face fiul... si il gaseste in camera cu bunica-sa, care avea fusta ridicata iar micul Bula era in spatele ei... la care taica-sau tipa:

- Bula, dar ce naiba faci acolo?

Iar Bulisor ii raspunde:

- Nu mai e asa de amuzant cand e vorba de maica-ta, nu-i asa?

*Intr-o zi, Bula trebău să meargă prin bucătărie. O clipă de neatentie la tocăt zărișuri și singele începe să curgă.

Îl repeză o cirpă, infasoară degetul și îl trăgează la dispensar. Intră în dispensar și vede două usi: "Bolnavi" și "Accidentati".

Că un cetățean disciplinat ce este, Bula intră la "Accidentati".

Acolo, ce să vezi? Alte două usi: "Cu sangerare" și "Fără sangerare".

Asa cum era firesc intră la "Cu sangerare". Alte două usi: "De importanță vitală" și "Fără importanță vitală"

Deschide usa "Fără importanță vitală" și se trezește îar în stradă...

Pleacă nervos acasă unde familia îl întreabă:

- Ei, Bula, te-ai tratat bine?
- Nu să uită nici dracu la mine, dar organizarea e la mare meserie.

**Bula se duce cu taica-să sa petească o fata. Ajunge la casa viitorului soțiu și Bula începe:

-Stimate domn, am venit să cer păzda fetei dumneavoastră!

Bula senior se întoarce scăzând:

-Mana, boule! Mana!

La care Bula, scos din minti:

-Cum tata? Să de acum încolo tot cu mana?

***Doi ardeleni:

- Ba, tu stii ce am patit ieri?
- Ce ai patit?
- Mergeam prin parc și vad că se apropiă o femeie tanără și frumusețea pe o bicicletă. Când a ajuns în drept cu mine, aruncă bicicleta în tufă și își smulge hainele de pe ea.
- Să...?
- Să-mi spune: - Iată ce trebuie!
- Să ce ai facut?
- Am luat bicicleta.
- Bine ai facut, hainele nu să le poată fi potrivite.

STRATEGIE_BANCURI.py

```
from abc import ABCMeta, abstractmethod

import re

from random import choice


class StrategieBancuri(metaclass=ABCMeta):

    def __init__(self):

        text = open("Bancuri.txt", 'r').read() # Citeste continutul fisierului "Bancuri.txt"

        self._dictionar = {"*": [], "**": [], "***": []} # Initializeaza un dictionar pentru stocarea bancurilor

        self._dictionar["*"] += re.findall("(?<!*\{2})(?<!*\{3})(?=<*)[^*]+", text) # Extrage bancurile cu o
        stea din text

        self._dictionar["**"] += re.findall("(?<!*\{3})(?=<*\{2})[^*]+", text) # Extrage bancurile cu doua
        stele din text

        self._dictionar["***"] += re.findall("(?=<*\{3})[^*]+", text) # Extrage bancurile cu trei stele din text

    @abstractmethod

    def get_banc(self):

        pass


class StrategieBancuri1Stea(StrategieBancuri):

    def __init__(self):

        super().__init__()

    def get_banc(self):

        return choice(self._dictionar["*"]) # Returneaza un banc cu o stea ales aleatoriu
```

```
class StrategieBancuri2Stele(StrategieBancuri):  
    def __init__(self):  
        super().__init__()  
  
    def get_banc(self):  
        return choice(self._dictionar["**"]) # Returneaza un banc cu doua stele ales aleatoriu
```

```
class StrategieBancuri3Stele(StrategieBancuri):  
    def __init__(self):  
        super().__init__()  
  
    def get_banc(self):  
        return choice(self._dictionar["***"]) # Returneaza un banc cu trei stele ales aleatoriu
```

STUDENT.py

```
import StrategieBancuri  
from random import choice  
  
class Student:  
    def __init__(self, name):  
        self.__name = name  
        self.__dict_reaction_to_banc_strategy = {}  
        self.__dict_reaction_to_banc_strategy["Buna"] = StrategieBancuri.StrategieBancuri1Stea() #  
        Initializează strategia pentru reacția "Buna"
```

```

        self.__dict_reaction_to_banc_strategy["Foarte buna"] = StrategieBancuri.StrategieBancuri2Stele() #
    Inițializează strategia pentru reactia "Foarte buna"

        self.__dict_reaction_to_banc_strategy["Cea mai buna"] = StrategieBancuri.StrategieBancuri3Stele()
    # Inițializează strategia pentru reactia "Cea mai buna"

        self.__possible_reactions = ["Buna", "Foarte buna", "Cea mai buna"] # Lista cu reacțiile posibile

def get_reaction(self):

    reaction = choice(self.__possible_reactions) # Alege o reacție aleatoriu din lista de reacții posibile
    print(f"Studentul {self.__name} are reactia: \"{reaction}\".")

    return reaction

def say_banc_to(self, other):

    reaction = other.get_reaction() # Obține reacția celuilalt student
    print(f"Studentul {self.__name} a primit reactia \"{reaction}\" și spune urmatorul banc:")

    print(self.__dict_reaction_to_banc_strategy[reaction].get_banc()) # Obține un banc în funcție de
    reacția primită

```

MAIN.py

```

import Student

if __name__ == '__main__':
    s1 = Student.Student("alex")
    s2 = Student.Student("gigi")
    s1.say_banc_to(s2)
    s2.say_banc_to(s1)

```

se creează în Python un program care primește un sir de căutare și mai multe nume de fișiere text (inclusiv sursă program). Aplicația caută dacă acest sir se găsește, apoi afișează pentru fiecare fișier numerele liniilor unde se află acesta. Programul primește de la linia de comandă parametrii de intrare, iar dacă este nevoie, mai primește și un sir de înlocuire a celui căutat, caz în care face search & replace.

Sa se creeze in Python un program care primește un sir de căutarea si mai multe nume de fisiere text(inclusiv sursa program).Aplicatia cauta daca acest sir se gaseste, apoi afiseaza pentru fiecare fisier numerele liniilor unde se afla acesta.Programul primeste de la linia de coamnda parametrii de intrare,iar daca este nevoie , mai primeste si un sir de inlocuire a celui cautat,caz in care facem search si replace

FILE1.TXT

will

mastermind

love

accumulation

sensitivity

fossil

switch

refer

love

density

effect

drown

FILE2.TXT

love
building
program
dance
entitlement
develop
reason
patient
chorus
fountain
love
window

MAIN.py

```
from functional import seq
import sys

if __name__ == '__main__':
    args = sys.argv[1:] # Obține argumentele de la linia de comandă
    i = 0
    if len(sys.argv) < 3 or sys.argv[1][0] != '-':
        # Verifică dacă există suficiente argumente sau dacă primul argument nu începe cu "-"
```

```
print("Parametri invalizi!\nParam1: -sir_de_cautat\nParam2: -sir_care_inlocuieste cel cautat(poate lipsi)\nRestul:file in care se face cautare")\n\n    sys.exit(1)\n\nfiles = seq(args).filter(lambda it: it[0] != '-').list()\n\n# Filtrăm argumentele care nu încep cu "-" pentru a obține lista de fișiere\n\nfor file in files:\n\n    lines = open(file, 'r').readlines() # Citim liniile din fișier\n\n    indexes = []\n\n    for index, line in enumerate(lines):\n\n        if line.count(args[0][1:]) > 0:\n\n            indexes.append(index)\n\n    print(f"File: {file}, lines with string: {indexes}")\n\n    # Afisăm numele fișierului și liniile care conțin sirul căutat și indexurile lor\n\n    if len(indexes) > 0 and args[1][0] == '-':\n\n        newLines = []\n\n        for line in lines:\n\n            newLines.append(line.replace(args[0][1:], args[1][1:]))\n\n        opened = open(file, 'w')\n\n        opened.writelines(newLines) # Înlocuim sirul căutat cu sirul de înlocuit și scriem liniile în fișier\n\n        opened.close()
```

Simulare de trimitere examen
PP/SD - Negru Parascheva, ☆
1211A Inbox

E

examenppsd@gmail... 11:18

to me ▾



se implementeze în Python echivalentul unei grupări de fire (ThreadPool) utilizând modulul threading și care să ne pună la dispoziție utilizând modelul comandă posibilitatea de a inspecta starea firului de execuție, de a-l opri temporar sau definitiv sau de a-l porni.

Sa se implementeze in Python echivalentul unei grupari de fire (ThreadPool)utilizand modulul threading si care sa ne puna la dispozitie utilizand modelul comanda posibilitatea de a inspecta starea firului de executie ,de a l opri temporarr sau definitiv sau de a l porni.

MAIN.py

```
import threading
import time
from time import sleep
from copy import copy
```

```
class MyThreadPool:
```

```

def __init__(self, n: int):
    self.__threads = [threading.Thread() for _ in range(n)] # Inițializează o listă de fire de execuție
    (thread-uri) goale

    self.__get_thread_state = lambda threads, i: threads[i].is_alive() # Funcție care returnează starea
    unui fir de execuție

    self.__delay_thread = lambda threads, i, delay: None # Funcție pentru întârzierea unui fir de
    execuție (nu face nimic)

    self.__start_thread = lambda threads, i: threads[i].start() # Funcție pentru pornirea unui fir de
    execuție

    self.__map = lambda func, *args: threading.Thread(target=func, args=args) # Funcție pentru crearea
    unui fir de execuție care să execute o funcție cu argumentele specificate

def __enter__(self):
    return self

def __exit__(self, exc_type, exc_value, tb):
    for thread in self.__threads:
        if thread.is_alive():
            thread.join() # Așteaptă finalizarea firelor de execuție active

def start_thread(self, i):
    self.__start_thread(self.__threads, i) # Pornește firul de execuție cu indicele specificat

def map(self, i, func, *args):
    if self.__threads[i].is_alive():
        self.__threads[i].join() # Așteaptă finalizarea firului de execuție cu indicele specificat, dacă este
        activ

        self.__threads[i] = self.__map(func, args) # Creaază un fir de execuție nou care să execute funcția cu
        argumentele specificate

def delay_thread(self, i, delay):

```

```
    self.__delay_thread(self.__threads, i, delay) # Întârzie firul de execuție cu indicele specificat pentru  
    un timp specificat

def get_thread_state(self, i):  
    return self.__get_thread_state(self.__threads, i) # Obține starea firului de execuție cu indicele  
    specificat

def print_lists(lists):  
    for list in lists:  
        for i in list:  
            print(f"{i} ", end="")  
            sleep(1) # Așteaptă 1 secundă între afișarea fiecărui element

def reverse_print_lists(lists):  
    for list in lists:  
        aux = copy(list)  
        aux.reverse() # Inversează lista  
        for i in aux:  
            print(f"{i} ", end="")  
            sleep(1) # Așteaptă 1 secundă între afișarea fiecărui element

if __name__ == '__main__':  
    with MyThreadPool(3) as pool: # Inițializează un obiect MyThreadPool cu 3 fire de execuție  
        list = [1, 2, 3, 4, 5]  
        pool.map(0, print_lists, list) # Asociază firului de execuție cu indicele 0 funcția print_lists cu lista  
        specificată  
        pool.start_thread(0) # Pornește firul de execuție cu indicele 0  
        print(f"Thread 0 is alive: {pool.get_thread_state(0)}") # Verifică starea firului de execuție cu indicele  
        0
```

```
pool.map(1, reverse_print_lists, list) # Asociază firului de execuție cu indicele 1 funcția  
reverse_print_lists cu lista specificată  
  
pool.start_thread(1) # Pornește firul de execuție cu indicele 1  
  
print(f"Thread 1 is alive: {pool.get_thread_state(1)}") # Verifică starea firului de execuție cu  
1  
  
time.sleep(7) # Așteaptă 7 secunde  
  
print(f"\nThread 0 is alive: {pool.get_thread_state(0)}") # Verifică starea firului de execuție cu  
indicele 0 după 7 secunde  
  
print(f"Thread 1 is alive: {pool.get_thread_state(1)}") # Verifică starea firului de execuție cu indicele  
1 după 7 secunde
```



Rnind de la exemplul cu `map_reduce` din modulul `more_itertools` (Python), să se calculeze indexul invers pentru un set de documente text (creat de student). Toate cele 3 funcții (`key_func`, `value_func` și `reduce_func`) vor fi scrise ca expresii lambda. Se va citi conținutul fiecărui document și apoi se va sparge și reuni într-o singură listă de cuvinte. Funcția de mapare va returna perechi de forma `<word, (document_id, 1)>`, în care cheia este cuvântul, iar valoarea este formată din tupla `(document_id (numele documentului), și 1 (o apariție a cuvântului în document))`. Funcția de reducere primește toate perechile, emite o pereche de forma `<word, [(document_id_i, count_word_in_document_id_i), (...), ...]>`, unde al doilea element din tuplă reprezintă numărul de aparții ale cuvântului în documentul respectiv. Pentru simplitate, funcția de reducere poate utiliza funcția `reduce_by_key` din biblioteca `PyFunctional`, ca să creeze acea listă de tuple. Exemplu de output:

```
{'a': [('text/1.txt', 1), ('text/2.txt', 2)],  
 'about': [('text/3.txt', 1)],  
 'always': [('text/3.txt', 1)],  
 'are': [('text/3.txt', 2)],  
 ...}
```

Pornind de la exemplul cu `map_reduce` din modulul `more_itertools` (Python), să se calculeze indexul invers pentru un set de documente text (creat de student). Toate cele 3 funcții (`key_func`, `value_func` și

reduce_func) vor fi scrise ca expresii lamnda. Se va citi continutul fiecarui document si apoi se va sparge si reuni intr o singura lista de cuvinte. Functie de mapare va returna perechi de forma <word , (document_id,1)>, in care cheia este cuvantul, iar valoarea este formata din tupla document_id (numele documentului) si 1 (o aparitie a cuvantului in document). Functia de reducere primeste toate perechile, emite o perete de forma , unde al doilea element din tupla reprezinta numarul de aparitii al cuvantului in documentul respectiv. Pentru simplitatea, functia de reducere poate utiliza functia reduce_by_key din biblioteca PyFunctional, ca sa creeze acea lista de tuple. Exemplu de output: {'a':...}

FILE1.TXT

will mastermind love accumulation sensitivity fossil switch refer love density effect drown

FILE2.TXT

love building program dance entitlement develop reason patient chorus fountain love window

MAIN.py

```
from more_itertools import flatten, map_reduce
from functional import seq

def read_words(files):
    words = []
    for file in files:
        aux = open(file).read().split(' ') # Citește conținutul fișierului și îl divide în cuvinte separate
```

```

aux = [(word, file) for word in aux] # Creează o pereche (cuvânt, fișier) pentru fiecare cuvânt din
fișierul respectiv

words.append(aux) # Adaugă perechile la lista de cuvinte

words = list(flatten(words)) # Aplatizează lista de liste de cuvinte

return words

if __name__ == '__main__':
    files = ["file1.txt", "file2.txt"] # Lista de fișiere

    words = read_words(files) # Citirea cuvintelor din fișiere

    key_func = lambda pair: pair[0] # Funcție pentru extragerea cheii (primul element al perechii)

    value_func = lambda pair: (pair[1], 1) # Funcție pentru extragerea valorii (al doilea element al
    perechii) și adăugarea valorii 1

    reduce_func = lambda list_of_pairs: seq(list_of_pairs).reduce_by_key(lambda x, y: x + y) # Funcție
    pentru reducerea valorilor cu aceeași cheie prin adunare

    dict = dict(map_reduce(words, keyfunc=key_func, valuefunc=value_func, reducefunc=reduce_func)) # Aplicarea operației de map-reduce pe lista de cuvinte pentru a obține un dicționar cu cuvintele și numărul de apariții

    print('{')

    for pair in dict.items(): # Parcurgerea perechilor cheie-valoare din dicționar
        print(f"\'{pair[0]}\' : {pair[1]}") # Afisarea cheii și valorii

    print('}')

```

i = 0 # Variabilă i care nu este utilizată în acest cod



examenppsd@gmail.com

to me ▾

scriu câte 100 de valori alese aleator în două fișiere CSV. Apoi aceste fișiere sunt deschise (se va utiliza o corutină) și informațiile sunt depuse în două colecții X și Y. Să se scrie un program Kotlin care utilizând funcții pure sau numai calcul lambda va găsi (dacă există) perechile de numere care satisfac $x * y = x + y * 3$.

Scriu cate 100 de valori alese aleator in doua fisere CSV.Apoi aceste fisiere sunt deschise (se va utiliza o corutina)si informatiile sunt depuse in doua colectii X si Y.Sa se scrie un program Kotlin care utilizand functii pure sau numai calcul lambda va gasi(daca exista)perechile de numere care satisfac $x*y=x+y*3$.

MAIN.py

```
import kotlinx.coroutines.async

import kotlinx.coroutines.runBlocking

import java.io.File

// Funcția `create_collections` primește o listă de nume de fișiere și returnează o listă de liste de întregi
// Aceasta citeste conținutul fiecărui fișier, îl separă după caracterul ',' și convertește fiecare element într-un întreg

fun create_collections(files: List<String>): List<List<Int>> {
    return files.map { File(it).readText().split(',').map { str -> str.toInt() } }
}
```

```

// Funcția `get_list_of_pairs` primește două liste de întregi, X și Y, și returnează o listă de perechi (x, y)

// Aceasta combină fiecare element din X cu fiecare element din Y și filtrează perechile care satisfac
condiția  $x * y == x + y * 3$ 

fun get_list_of_pairs(X: List<Int>, Y: List<Int>): List<Pair<Int, Int>> {

    return X.flatMap { x -> Y.filter { y -> x * y == (x + y * 3) }.map { y -> Pair(x, y) } }
}

fun main() = runBlocking {

    val files = listOf("file1.csv", "file2.csv") // Lista de nume de fișiere

    val rez1 = async { create_collections(files) } // Apelarea asincronă a funcției `create_collections` pentru
    a obține rezultatul în mod asincron

    val X = rez1.await()[0] // Extrage prima listă din rezultatul asincron al funcției `create_collections` și o
    atribuie lui X

    val Y = rez1.await()[1] // Extrage a doua listă din rezultatul asincron al funcției `create_collections` și o
    atribuie lui Y

    println("X: $X\nY: $Y") // Afisează listele X și Y

    val rez2 = async { get_list_of_pairs(X, Y) } // Apelarea asincronă a funcției `get_list_of_pairs` pentru a
    obține rezultatul în mod asincron

    val list_of_pairs = rez2.await() // Așteaptă finalizarea rezultatului asincron și îl atribuie lui list_of_pairs

    println("Perechi (x, y) care satisfac  $x * y == x + y * 3$ : $list_of_pairs") // Afisează lista de perechi (x, y)
    care satisfac condiția specificată
}

```

Găsiți mesaje, documente, fotografii sau persoane



← Înapoi



Arhivare

Mutare

Ștergere

Spam

...



Kot-02

Să se scrie un program Kotlin utilizând corutine și canale "conflated" pentru comunicarea între procese care distribuie într-o manieră ciclică (cuvânt1 - proces1, cuvânt2 - proces2, cuvânt3 - proces3, cuvânt 4 - proces1), câte un cuvânt citit dintr-un fișier către alte trei procese, iar fiecare din acestea afișează numele lui și cuvântul primit la consolă.

bafta

Kot-02

Sa se scrie un program Kotlin utilizând coruline si canale "conflated" pentru comunicarea intre procese care distribuie intr o maniera ciclica(cuvant1-proces1,cuvant2-proces2,cuvant3-proces3,cuvant4-proces1),cate un cuvant citit dintr un fisier catre alte trei procese,iar fiecare din acestea afiseaza numele lui si cuvantul primit la consola.

MAIN.py

```
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.channels.Channel
import kotlinx.coroutines.channels.ClosedReceiveChannelException
import kotlinx.coroutines.launch
import kotlinx.coroutines.runBlocking
import java.io.File

// Funcția `get_from_channel_then_send_to_another` primește un canal de tip String și realizează transferul de date între canalele interne.

// Se creează trei canale interne de tip String. Se lansează mai multe corutine pentru a apela funcția `print_string` pentru fiecare canal intern.
```

```
// Apoi, într-un loop infinit, se primește un element din canalul de intrare și se trimite la unul dintre canalele interne, într-o rotație ciclică.

// Se închid canalele interne și se așteaptă finalizarea corutinelor.

suspend fun CoroutineScope.get_from_channel_then_send_to_another(ch: Channel<String>) {

    val ch_list = listOf(Channel<String>(), Channel<String>(Channel.CONFLATED),
    Channel<String>(Channel.CONFLATED))

    val tasks = ch_list.map { channel -> launch { print_string(channel) } }

    var i = 0

    try {

        while (true) {

            ch_list[i].send(ch.receive())

            i = (i + 1) % 3

        }

    } catch (e: ClosedReceiveChannelException) {

        ch_list.forEach { it.close() }

        tasks.forEach { it.join() }

    }

}
```

```
// Funcția `print_string` primește un canal de tip String și afișează elementele primite din canal până la închiderea canalului.

suspend fun print_string(ch: Channel<String>) {

    try {

        while (true)

            println("Thread \\"${Thread.currentThread()}\\" -> \"${ch.receive()}\\"")

    } catch (e: ClosedReceiveChannelException) {

        // Canalul a fost închis, nu mai avem elemente de citit

    }

}
```

```
fun main() = runBlocking {  
    val ch_list = listOf(Channel<String>(), Channel<String>(Channel.CONFLATED),  
    Channel<String>(Channel.CONFLATED))  
  
    val words = ArrayList<String>(File("text.txt").readText().split(' '))  
  
    val tasks = ch_list.map { channel -> launch { get_from_channel_then_send_to_another(channel) } }  
  
    var i = 0  
  
    while (!words.isEmpty()) {  
        ch_list[i].send(words.removeFirst())  
  
        i = (i + 1) % 3  
  
    }  
  
    ch_list.forEach { it.close() }  
  
    tasks.forEach { it.join() }  
}
```

Simulare de trimitere examen PP/ SD - Miron Raul-Emilian, 1206A ➤ ☆

Mesaje primite



examenppsd 11:16

către mine ▾

◀ ...

ajutorul modelului memento să se creeze un program Python care va permite aplicarea succesivă a unor lambda funcții peste o colecție inițializată cu date citite dintr-un fișier și apoi va restaura obiectele la starea anterioară în momentul în care acest lucru este cerut (la consolă). Se vor aplica următoarele funcții:
-> $f1(x)=x+1$ (dacă x par), altfel $f1(x)=x$;
-> $f2(x)=3x^2-2x+1$;
-> $f3(x)=x_i+x_{i+1}$ unde i este indexul x-ului primit ca parametru
Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

Ajutorul modelului memento sa se creeze un program Python care va permite aplicarea succesiva a unor lambda functii peste o colectie initializata cu date citite dintr un fisier si apoi va restaura obiectele la starea anterioara in momentul in care acest lucru este cerut (la consola).Se vor aplica urmatoarele functii:-> $f1(x)=x+1$ (daca x par),altfel $f1(x)=x$;

-> $f2(x)=3x^2-2x+1$;

->f3(x)=xi+xi+1 unde i este indexul x-ului primit ca parametrul.Se vor desena diagrame de clase si de obiecte.Se va explica maniera de aplicarea a principiilor SOLID.

MEMENTO.py

```
class Originator:  
    def __init__(self, l):  
        self.__state = l  
  
    def set_state(self, l):  
        self.__state = l  
  
    def get_state(self):  
        return self.__state  
  
    def save_state_to_memento(self):  
        return Memento(self.__state)  
  
    def restore_state_from_memento(self, memento):  
        self.__state = memento.get_state()  
  
  
class Memento:  
    def __init__(self, state):  
        self.__state = state  
  
    def get_state(self):  
        return self.__state
```

```
class Caretaker:  
  
    def __init__(self):  
        self.__states = []  
  
  
    def add(self, state):  
        self.__states.append(state)  
  
  
    def get(self):  
        rez = self.__states[-1]  
        self.__states.remove(self.__states[-1])  
        return rez
```

MAIN.py

```
from Memento import *  
  
from functional import seq  
  
  
if __name__ == '__main__':  
    numbers=open("file1.csv",'r').read().split(',')  
    numbers=seq(numbers).map(lambda it:int(it)).list()  
  
  
f1=lambda x:(x%2==0 and x+1) or x
```

```
f2=lambda x:3*x*x-2*x+1
f3=lambda x:x+x+1
functions=[f1,f2,f3]

caretaker=Caretaker()
originator=Originator(numbers)

print(f"Lista originala: {numbers}")
for i,f in enumerate(functions,1):
    caretaker.add(originator.save_state_to_memento())
    originator.set_state(seq(originator.get_state()).map(f).list())
    print(f"Lista dupa aplicarea functiei f{i}: {originator.get_state()}")
restore=input("Doriti restaurarea starii anterioare a listei? 1-Da Altceva-Nu\nRaspunsul dumneavostra: ")
if restore=="Da":
    originator.restore_state_from_memento(caretaker.get())
print(f"Lista finala: {originator.get_state()}")
```

ilizând sevențe din Python (PyFunctional), corutinele (asyncio) și împărțirea în mai multe bucăți, să se eliminate dintr-un fișier text rezultat din conversia unui epub (aveți aşa ceva de la o temă) spațiile multiple, salturile la linie nouă. La sfârșit va fi generat noul fișier. Se vor folosi abordări de tipul împachetează-procesează-despachetează.



utilizand sevante din Python(PyFunctional),crutinele(asyncio)si impartirea in mai multe bucati, sa se eliminate dintr un fisier text rezultat din conversia unui epub(aveti asa ceva de la o tema)spatiile multiple,salturile la linie noua.La sfrasit va fi generat noul fiser.Se vor folosi abordari de tipul impacheteaza-proceseaza-despacheteaza.

MAIN.py

```
from functional import seq
import asyncio
import re

coroutines_per_processing = 4

async def _elimin_spatii_multiple(partition: str):
    return re.sub("[ ]{2,}", ' ', partition)
```

```
async def elimin_spatii_multiple(text):

    # text = seq(open(file, 'r').readlines())

    partition_len = text.len() // coroutines_per_processing + 1

    partitions = []

    while text.len() > 0:

        partitions.append(text.take(partition_len).make_string(''))

        text = text.drop(partition_len)

    tasks = [asyncio.create_task(_elimin_spatii_multiple(partition)) for partition in partitions]

    new_text = ".join([await task for task in tasks])

    return new_text
```

```
async def _elimin_salturi_linie_noua(partition: str):

    return partition.replace('\n', '')
```

```
async def elimin_salturi_linie_noua(text):

    partition_len = text.len() // coroutines_per_processing + 1

    partitions = []

    while text.len() > 0:

        partitions.append(text.take(partition_len).make_string(''))

        text = text.drop(partition_len)

    tasks = [asyncio.create_task(_elimin_salturi_linie_noua(partition)) for partition in partitions]

    new_text = ".join([await task for task in tasks])

    return new_text
```

```
async def main():
```

```
file = "economics_to_be_happier.txt"
text = seq(open(file, 'r').readlines())
task = asyncio.create_task(elimin_spatii_multiple(text))
new_text = await task
print(f"Dupa eliminarea spatilor multiple:\n{new_text}")
task = asyncio.create_task(elimin_salturi_linie_noua(seq(new_text)))
new_text = await task
print(f"Dupa eliminarea salturilor la linie noua:\n{new_text}")
with open("OUTPUT.txt", 'w') as output:
    output.write(new_text)

asyncio.run(main())
```

Simulare de trimitere examen

PP/SD – Mihalache Nicoleta-

Ecaterina, 12113 Mesaje primite



E

examenppsd@gmail.com

11:18

către mine ▾



ilizând modelul adaptor să se creeze un program Kotlin care va primi

la intrare o listă cu mai multe fișiere (care pot fi de diverse tipuri

de date simple sau de tip colecție) și va realiza utilizând câte o corutină pentru fiecare fișier scrierea datelor într-un fișier destinație unic (ordinea de scriere a datelor nu contează). Se vor

desena diagrama de clase și de obiecte. Se va explica maniera de

aplicare a principiilor SOLID.

Utilizand modelul adaptor sa se creeze un program Kotlin care va primi la intrare o lista cu mai multe fisiere(care pot fi de diverse tipuri de date simple sau de tip colectie)si va realiza utilizand cate o corutina pentru fiecare fisier scrierea datelor intr un fisier destinatie unic(ordinea de scriere a datelor nu conteaza).Se vor desena diagramea de clase si de obiecte.Se va explica maniera de aplicarea a principiilor SOLID.

ADAPTER.KT

```
import kotlinx.coroutines.coroutineScope  
import kotlinx.coroutines.launch
```

```
import java.util.concurrent.locks.ReentrantLock

class Adapter(lock:ReentrantLock, files>List<String>,
output_file:String):ConcurrentFileWriter(lock) {
    private val readers_and_writers=files.map {
ReaderAndWriter(it,output_file) }
    override suspend fun write()= coroutineScope {
        val tasks=readers_and_writers.map { launch {
lock.lock();it.write(it.read());lock.unlock() } }
        tasks.forEach { it.join() }
    }
}
```

READERANDWRITER.KT

```
import java.io.File
import java.io.FileWriter

class ReaderAndWriter(private val file:String, private val
output_file:String) {
    fun read():String{
        return File(file).readText()
    }
    fun write(output: String){
        val fw=FileWriter(output_file,true)
        fw.write(output)
        fw.close()
    }
}
```

CONCURRENTFILEWRITER.KT

```
import java.util.concurrent.locks.ReentrantLock

abstract class ConcurrentFileWriter(protected val lock:ReentrantLock) {
    abstract suspend fun write()
}
```

MAIN.KT

```
import kotlinx.coroutines.runBlocking
import java.util.concurrent.locks.ReentrantLock

fun main()= runBlocking{
    val files=listOf("file1.csv","file2.csv","file3.csv","file4.csv")
    val output_file="OUTPUT.txt"
    val adapter=Adapter(ReentrantLock(),files,output_file)
    adapter.write()
}
```

Simulare de trimitere examen
PP/SD - Para Ramona-Maria, ☆
1208A



examenppsd@gmail... 11:17

către mine ▾



ntr-un fișier text cu minim 50 de valori (numere) se iau
câte 10
valori consecutive care vor fi procesate într-un thread
separat (din
modulul threading, Python) (deci se lansează în
execuție minim 5
thread-uri). Datele sunt depuse într-un ADT, apoi se va
efectua o
procesare de tip lambda care va găsi minimul,
maximul și media din
secvență, apoi va extrage o submulțime ce conține
numai numerele care
se află în intervalul media + sau - media pătratică a
secvenței
procesate.

Intr un fisier text cu minim 50 de valori(numere)se iau cate 10 valori consecutive care vor fi procesate
intr un thread separat(dint modulul threading,Python)(deci se lanseaza in executie minim 5 thread-
uri)Datele sunt depuse intr un ADT, apoi se va efectua o procesare de tip lambda care va gasi
minimul,maximul si media din secventa,apoi va extrage o submultime ce contine numai numerele care
se afla in intervalul media+sau -media patratica a secventei procesate.

VALORI.TXT

```
74 61 36 9 74 79 69 60 7 80 69 1 55      64 59 36 31 78 25 48 99 21 30 35 2 76 78 50 91 71 31 64 98 3 6  
6 10 68 21 52 70 8 63 25 23 27 9 26 85 76
```

MAIN.PY

```
from math import sqrt  
  
from functional import seq  
  
from threading import Thread  
  
  
min_fun = lambda list: seq(list).min()  
  
max_fun = lambda list: seq(list).max()  
  
mean_fun = lambda list: seq(list).sum() / seq(list).len()  
  
square_mean_fun = lambda list: sqrt(seq(list).map(lambda it: it * it).sum() / seq(list).len())  
  
subsequence_fun = lambda list: seq(list).filter(lambda it: -square_mean_fun(list) <= it <= square_mean_fun(list))  
  
if __name__ == "__main__":  
  
    lists = []  
  
    numbers = seq(open("Valori.txt", "r").readline().split(' ')) \  
        .map(lambda it: it.replace('\t', '')) \  
        .map(lambda it: int(it))  
  
    while numbers.len() > 0:  
  
        if numbers.len() >= 10:  
  
            lists.append(numbers.take(10).list())  
  
            numbers = numbers.drop(10)  
  
        else:  
  
            lists.append(numbers.take(numbers.len()).list())  
  
            numbers = numbers.drop(numbers.len())
```

```

results = [] # min,max,mean,subsequence
threads = []
for list in lists:
    aux = Thread(target=lambda list, results: results.append(
        (min_fun(list), max_fun(list), mean_fun(list), subsequence_fun(list))), args=(list, results,))
    aux.start()
    threads.append(aux)

for thread in threads:
    thread.join()

i=0
for i in range(len(lists)):
    print(f"Sequence:{lists[i]}, min:{results[i][0]}, max:{results[i][1]}, mean:{results[i][2]},\nsubsequence:{results[i][3]}")

```

MAIN2.PY

```

from math import sqrt
from functional import seq
import threading

min_fun = lambda list: seq(list).min()
max_fun = lambda list: seq(list).max()
mean_fun = lambda list: seq(list).sum() / seq(list).len()
square_mean_fun = lambda list: sqrt(seq(list).map(lambda it: it * it).sum() / seq(list).len())

```

```

subsequence_fun = lambda list: seq(list).filter(lambda it: mean_fun(list)-square_mean_fun(list) <= it <=
mean_fun(list)+square_mean_fun(list))

if __name__ == "__main__":
    lists = []
    numbers = seq(open("Valori.txt", "r").readline().split(' '))
    .map(lambda it: it.replace('\t', ''))
    .map(lambda it: int(it))

    while numbers.len() > 0:
        if(numbers.len() >= 10):
            lists.append(numbers.take(10).list())
            numbers = numbers.drop(10)
        else:
            lists.append(numbers.take(numbers.len()).list())
            numbers = numbers.drop(numbers.len())

    results = []
    threads = []
    for list in lists:
        aux = threading.Thread(target=lambda list, results: results.append((min_fun(list), max_fun(list),
mean_fun(list), square_mean_fun(list) , subsequence_fun(list))), args=(list, results, ))
        aux.start()
        threads.append(aux)

    for thread in threads:
        thread.join()

i=0

```

```
for i in range(len(lists)):  
    print(f"Sequence:{lists[i]}, min:{results[i][0]}, max:{results[i][1]}, mean:{results[i][2]},  
square_mean_fun:{float(results[i][3])} subsequence:{results[i][4]}")  
i=0
```

**Simulare de trimitere examen
PP/SD - Dandu Adriana-Mihai,
1209B**

se scrie un program Python care va utiliza modelul mediator pentru a realiza o interacțiune de tipul transmitere bidirecțională (folosind cozi între procese) între studenți și profesori utilizând asistentul (un obiect profesor, unul asistent și mai multe obiecte student). Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

Se scrie un program Python care va utiliza modelul mediator pentru a realiza o interacțiune de tipul transmitere bidirecțională (folosind cozi între procese) între studenți și profesori utilizând asistentului (un obiect profesor, unul asistent și mai multe obiecte student). Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicarea a principiilor SOLID.

MEDIATOR.PY

```
from abc import ABCMeta, abstractmethod
```

```
class Mediator(metaclass=ABCMeta):
```

```
    def __init__(self):
```

```
self._profesor=None
self._studenti=None

@abstractmethod
def ask_if_passed(self, grade):
    pass

@abstractmethod
def ask_if_attending(self, i):
    pass

def set_studenti(self,studenti):
    self._studenti=studenti

def set_profesor(self,profesor):
    self._profesor=profesor

class Asistent(Mediator):
    def __init__(self):
        super().__init__()

    def ask_if_passed(self,grade):
        return super()._profesor.respond_if_passed(grade)

    def ask_if_attending(self,i):
        return super()._studenti[i].respond_if_attending()
```

STUDENT.PY

```
class Student:  
  
    def __init__(self, name, asistent):  
        self.__name = name  
        self.__grade = 0  
        self.__asistent = asistent  
        self.__passed = False  
        self.__attending=False  
  
    def do_exam(self, grade):  
        self.__grade = grade  
  
    def ask_if_passed(self):  
        self.__passed = self.__asistent.ask_if_passed(self.__grade)  
  
    def respond_if_attending(self):  
        return self.__attending  
  
    # def set_passed(self, passed):  
    #     self.__passed=passed
```

PROFESOR.PY

```
class Profesor:  
  
    def __init__(self, asistent):  
        self.__asistent=asistent  
  
    def respond_if_passed(self, grade):
```

```
return grade>=4.5

def ask_if_attending(self, i):
    if(self.__asistent.ask_if_attending(i)):
        print("Creating one more exam subject...")
```

MAIN.PY

```
from Student import Student
from Profesor import Profesor
from Mediator import Mediator, Asistent

if __name__ == '__main__':
    asistent=Asistent()
    studenti=[Student(name,asistent) for name in ["Alex","George","Alina"]]
    profesor=Profesor(asistent)
    asistent.set_profesor(profesor)
    asistent.set_studenti(studenti)
```

Pornind de la exemplul din curs cu functori none și some, să se creeze în Kotlin o transformare (functor) care se aplică asupra lui toInt. Astfel, dacă este none se va înlocui cu -1, dacă este some se generează o valoare aleatoare de zero sau unu). Apoi transformarea va fi aplicată cu map.

Pornind de la exemplul din curs cu functori none și some, să se creeze în Kotlin o transformare (functor) care se aplică asupra lui toInt. Astfel, dacă este none se va înlocui cu -1, dacă este some se generează o valoare aleatoare de zero sau unu). Apoi transformarea va fi aplicată cu map.

MAIN.KT

```
sealed class TipSimplu<out T>{
    object None:TipSimplu<Nothing>() {
        override fun toString()="Cu None"
    }
    data class Some<out T>(val value:T):TipSimplu<T>()
    companion object
}

fun <T>TipSimplu<T>.toInt():Int=when(this){
    TipSimplu.None->-1
    is TipSimplu.Some->(0..1).random()
}

fun TipSimplu<Int>.map(transform:(TipSimplu<Int>)->Int):TipSimplu<Int>{
    return TipSimplu.Some<Int>(transform(this))
}

fun main()
{
    println(TipSimplu.None.map(TipSimplu<Int>::toInt))
    println(TipSimplu.Some(3).map(TipSimplu<Int>::toInt))
    println(TipSimplu.Some(5).map(TipSimplu<Int>::toInt))
}
```

se implementeze în Python un automat cu trei stări unde fiecare stare este gestionată de un proces în care să se proceseze cu expresii lambda o listă de numere întregi trimisă ca argument în constructor astfel:

În s0 se verifică dacă mai sunt elemente în listă (dacă da, se duce în starea s1 unde se identifică și se șterge primul element par găsit, apoi se trece în s2).

În s2 se identifică și se șterge primul element impar găsit, apoi se trece în s0.

La fiecare ștergere se afișează elementul șters.

În s0 dacă nu mai există elemente, se anunță acest lucru, iar programul se termină.

Sa se implementeze in Python un automat cu trei stari unde fiecare stare este gestionata de un proces in care sa se proceseze cu expresii lambda o lista de numere intregi trimisa ca argument in constructor astfel:

In s0 se verifica daca mai sunt elemente in lista(daca da,se duce in starea s1 unde se identifica si se sterge primul elemnet par gasit,apoi se trece in s2).In s2 se identifica si se sterge primul elemente impar gasit, apoi se trece in s0.La fiecare stergere se afiseaza elementul sters.In s0 daca nu mai exista elemente, se anunta acest lucru,iar programul se termina.

MAIN.PY

```
import time
from multiprocessing import Process, Queue
```

```
from functional import seq
```

```
def State0(q):
```

```
    l = q.get()
```

```
    if len(l) > 0:
```

```
        q.put(True)
```

```
        q.put(l)
```

```
    else:
```

```
        q.put(False)
```

```
    time.sleep(1)
```

```
def State1(q):
```

```
    l: list = q.get()
```

```
    if seq(l).filter(lambda it:it%2==0).len()>0:
```

```
        aux = seq(l).filter(lambda it: it % 2 == 0).first()
```

```
        l.remove(aux)
```

```
        print(f"State 1 a eliminat {aux}")
```

```
    q.put(True)
```

```
    q.put(l)
```

```
# time.sleep(1)
```

```
def State2(q):
```

```
    l: list = q.get()
```

```
    if seq(l).filter(lambda it:it%2!=0).len()>0:
```

```

aux = seq(l).filter(lambda it: it % 2 != 0).first()

l.remove(aux)

print(f"State 2 a eliminat {aux}")

q.put(True)

q.put(l)

# time.sleep(1)

class FSM:

    def __init__(self, list):

        self.__queue = Queue()

        self.__queue.put(True)

        self.__queue.put(list)

        self.__states=[State0,State1,State2]

        self.__current_state = 0

    def start(self):

        while self.__queue.get():

            process = Process(target=self.__states[self.__current_state], args=(self.__queue,))

            process.start()

            self.__current_state = (self.__current_state + 1) % 3

            process.join()

if __name__ == "__main__":

    l = list(range(10))

    fsm = FSM(l)

    fsm.start()

```

Telekom Romania  **AD**

Comandă online Smart WiFi și ai livrare gratis...
Oriunde vrei să mergi, ai net instant la priză. ...

Simulare de trimitere examen PP/SD 
- Susanu Alexandru-Catalin, 1210B

 **examenppsd@gmail.com**
To alex_cata78@yahoo.com
Jun 27 at 11:18 AM

ilizând generice și funcții de extensie să se scrie în Kotlin un aplicative care aplică într-o corutină următoarea transformare lambda
{i -> i //3 } și { i -> i *5 } asupra elementelor dintr-o listă încărcată dintr-un fișier text de intrare. Rezultatul va fi convertit la string și afișat.

Utilizand generice si functii de extensie sa se scrie in Kotlin un aplicative care aplica intr o corutina urmatoarea transformarea lambda{i->i//3} si {i->i*5} asupra elementelor dintr o lista incarcata dintr un fisier text de intrare.Rezultatul va fi converitit la string si afisat.

```
import kotlinx.coroutines.async
import kotlinx.coroutines.runBlocking
import java.io.File

fun <T, R>List<T>.ap(fab:List<(T) ->R>):List<R> = fab.flatMap { this.map(it) }

fun main()= runBlocking {
```

```
val numbers= File("text.txt").readText().split(' ').map{ it.toInt() }
val functii=listOf<(Int)->Int>({i->i/3}, {i->i*5})
val task=async{ numbers.ap(functii) }
val newNumbers=task.await()
println(newNumbers)
}
```



Sa se proiecteze si sa implementeze o aplicatie care citeste dintr un fisier un text scris de student care contine si erori de scriere a unor cuvinte.Aceasta extrage si analizeaza fiecare cuvant, verificand un dictionar(separat dat de student cu cateva forme corecte de cuvinte).Totul se realizeaza intr o corutina.De asemenea, cu observer(similarr cu autocorect-ul din Word) declanseaza intrebarea in momentul in care intalnesc un cuvant gresit daca doresc corectarea cuvantului, iar cu memento permite sa faca undo peste corectarea cuvantului, iar cu memento permit sa fac undo peste corecturile deja efectuate.Se vor desena diagrama de clase si de obiecte.

CONCRETEOBSERVER.KT

```
import java.io.BufferedReader
import java.io.InputStreamReader
import java.util.*

class ConcreteObserver(subject: Subject, caretaker: Caretaker):
Observer(subject, caretaker)
{
    override fun update(word: String) {
        val correctedWord=dict[word]
        if(correctedWord != null){
            print("Cuvantul $word pare gresit. Varianta corecta ar putea fi
${dict[word]} + "
"\nDoriti corectarea lui? 1-Da 0-Nu\nRaspunsul
dumneavoastră: ")
```

```

if(BufferedReader(InputStreamReader(System.`in`)).readLine().toInt()==1)
{
    subject.addWord(correctedWord)
    caretaker.add(subject.saveState())
}
else subject.addWord(word)
else subject.addWord(word)
}
}

```

ORIGINATOR.KT

```

abstract class Originator{
    private var state= arrayListOf<String>()
    fun addWord(word:String){state.add(word) }
    fun saveState():Memento= Memento(Pair(state.size-1,state.last()))
    fun restoreState(memento:
Memento){state[memento.getState().first]=memento.getState().second}
    fun getWords():List<String>=state
}

```

SUBJECT.KT

```

class Subject:Originator() {
    protected val observers= arrayListOf<Observer>()
    fun register(observer:Observer){observers+=observer}
    fun unregister(observer: Observer){observers-=observer}
    fun notifyAll(word:String){observers.forEach{it.update(word) }}
}

```

CARETAKER.KT

```

class Caretaker {
    private val mementoList= arrayListOf<Memento>()
    fun add(memento:Memento){mementoList+=memento}
    fun getMemento(ind:Int):Memento{return mementoList.removeAt(ind) }
    fun getNumberOfMementos():Int=mementoList.size
}

```

OBSERVER.KT

```

abstract class Observer(protected val subject: Subject, protected val caretaker: Caretaker) {
    protected val dict=mapOf("wrk" to "work", "cod3" to "code", "cart" to "chart")
    abstract fun update(word: String)
}

```

MEMENTO.KT

```

class Memento(private val state:Pair<Int, String>) {
    fun getState()=state
}

```

MAIN.KT

```

import kotlinx.coroutines.launch
import kotlinx.coroutines.runBlocking
import java.io.BufferedReader
import java.io.File
import java.io.InputStreamReader

fun main()= runBlocking {
    val br=BufferedReader(InputStreamReader(System.`in`))
    val subject=Subject()
    val caretaker=Caretaker()
    var task=launch {
        val words= File("text.txt").readText().split(' ')
        val observer=ConcreteObserver(subject,caretaker)
        subject.register(observer)
        words.forEach { subject.notifyAll(it) }
    }
    task.join()
    task=launch {
        var ok = true
        while (ok && caretaker.getNumberOfMementos() > 0) {
            print("Se poate face undo la ${caretaker.getNumberOfMementos()}\n")
            corecturi.\n-1=Nu doresc, index_corectura=Indexul corecturii la care se vrea
            a se da undo\n Raspunsul dumneavoastra: ")
            val ind = br.readLine().toInt()

            if (ind !in 0 until caretaker.getNumberOfMementos())
                ok = false
            else subject.restoreState(caretaker.getMemento(ind))
        }
        println("Lista finala de cuvinte: ${subject.getWords() }")
    }
}

```

```
    task.join()  
}
```