**Практическая работа №4**

**Сапицкий Алексей К4110с**

**Задния сделаны к паттернам Адаптер, Абстрактная фабрика и Фабричный метод, выполнено на Swift**

**Паттерн Адаптер**

**import** UIKit

//MARK: **- Упражнение**

**protocol** Game {

**func** makeThrow() -> Int

}

**class** Dice: Game {

**func** makeThrow() -> Int { Int.random(in: 1...6) }

}

**class** Player {

**var** toString: String { name }

**var** name: String

**init**(**\_** name: String) {

**self**.name = name

}

**func** gameTry(game: Game) -> Int { game.makeThrow() }

}

**let** dice = Dice()

**let** player = Player("Alex")

print("The result of \(player.toString) is \(dice.makeThrow())")

**class** Coin {

**func** makeCoinThrow() -> Int { Int.random(in: 1...2) }

}

**class** GameAdapter: Game {

**private** **let** coin: Coin

**init**(**\_** coin: Coin) {

**self**.coin = coin

}

**func** makeThrow() -> Int { coin.makeCoinThrow() }

}

**let** coin = Coin()

**let** adapter = GameAdapter(coin)

print("The result of \(player.toString) is \(adapter.makeThrow())")

//MARK: **- Задание**

**struct** Celsius {

**var** toString: String { "\(value)" }

**let** value: Double

**init**(celsius: Double) {

**self**.value = celsius

}

**init**(farenheits: Farenheit) {

**self**.value = (farenheits.value - 32) \* 5/9

}

}

**struct** Farenheit {

**let** value: Double

}

**protocol** Sensor {

**func** getIndications() -> String

}

**class** ClimateSystem {

**private** **let** sensors: [Sensor]

**init**(**\_** sensors: [Sensor]) {

**self**.sensors = sensors

}

**func** showIndications() {

sensors.forEach{ print($0.getIndications()) }

}

}

**class** TemperatureSensor {

**func** getTemperature() -> Farenheit { Farenheit(value: Double.random(in: 0...10)) }

}

**class** SensorAdapter: Sensor {

**private** **let** sensor: TemperatureSensor

**init**(**\_** sensor: TemperatureSensor) {

**self**.sensor = sensor

}

**func** getIndications() -> String {

**let** temp = sensor.getTemperature()

**return** Celsius(farenheits: temp).toString

}

}

**let** tempSensor = TemperatureSensor()

**let** tempAdapter = SensorAdapter(tempSensor)

**let** system = ClimateSystem([tempAdapter])

system.showIndications()

**Паттерн Абстрактная фабрика**

**import** UIKit

//MARK: **- Упражнение**

**protocol** Car {

**var** name: String { **get** **set** }

**var** body: String { **get** **set** }

**func** maxSpeed(engine: Engine) -> Int

}

**protocol** Engine {

**var** maxSpeed: Int { **get** }

}

**protocol** CarFactory {

**func** createCar() -> Car

**func** createEngine() -> Engine

}

**class** Ford: Car {

**var** body: String = "Muscle"

**var** name: String = "Ford"

**func** maxSpeed(engine: Engine) -> Int { engine.maxSpeed }

}

**class** FordEngine: Engine {

**var** maxSpeed: Int = 220

}

**class** FordFactory: CarFactory {

**func** createCar() -> Car { Ford() }

**func** createEngine() -> Engine { FordEngine() }

}

**class** Client {

**private** **let** car: Car

**private** **let** engine: Engine

**var** toString: String { car.name }

**var** maxSpeed: String { "\(engine.maxSpeed)" }

**init**(**\_** factory: CarFactory) {

**self**.car = factory.createCar()

**self**.engine = factory.createEngine()

}

}

**let** factory: CarFactory = FordFactory()

**let** client = Client(factory)

print("Максимальная скорость - \(client.toString) составляет \(client.maxSpeed) км/ч")

//MARK: **- Задание**

**struct** Audi: Car {

**var** body: String = "Coupe"

**var** name: String = "Audi"

**func** maxSpeed(engine: Engine) -> Int { engine.maxSpeed }

}

**struct** AudiEngine: Engine {

**var** maxSpeed: Int = 250

}

**class** AudiFactory: CarFactory {

**func** createCar() -> Car { Audi() }

**func** createEngine() -> Engine { AudiEngine() }

}

**let** factoryAudi: CarFactory = AudiFactory()

**let** client2 = Client(factoryAudi)

print("Максимальная скорость - \(client2.toString) составляет \(client2.maxSpeed) км/ч")

**Паттерн Цепочка обязанностей**

**import** UIKit

**class** Receiver {

**var** bankTransfer: Bool

**var** moneyTransfer: Bool

**var** paypalTransfer: Bool

**init**(isBank: Bool, isMoney: Bool, isPayPal: Bool) {

**self**.bankTransfer = isBank

**self**.moneyTransfer = isMoney

**self**.paypalTransfer = isPayPal

}

}

**protocol** PaymentHandler {

**var** successor: PaymentHandler? { **get** **set** }

**func** handle(receiver: Receiver)

}

**class** BankPaymentHandler: PaymentHandler {

**var** successor: PaymentHandler?

**func** handle(receiver: Receiver) {

**if** receiver.bankTransfer {

print("Bank transfering...")

} **else** {

successor?.handle(receiver: receiver)

}

}

}

**class** MoneyPaymentHandler: PaymentHandler {

**var** successor: PaymentHandler?

**func** handle(receiver: Receiver) {

**if** receiver.moneyTransfer {

print("Transfering money")

} **else** {

successor?.handle(receiver: receiver)

}

}

}

**class** PayPalPaymentHandler: PaymentHandler {

**var** successor: PaymentHandler?

**func** handle(receiver: Receiver) {

**if** receiver.paypalTransfer {

print("Transfering thru paypal")

} **else** {

successor?.handle(receiver: receiver)

}

}

}

**let** receiver = Receiver(isBank: **false** , isMoney: **true**, isPayPal: **true**)

**var** bankHandler = BankPaymentHandler()

**var** moneyHandler = MoneyPaymentHandler()

**var** paypalHandler = PayPalPaymentHandler()

bankHandler.successor = moneyHandler

moneyHandler.successor = paypalHandler

bankHandler.handle(receiver: receiver)

**Паттерн Команда**

**import** UIKit

**protocol** Command {

**var** operand: Double { **get** **set** }

**var** unit: ArithmeticUnit { **get** **set** }

**func** execute()

**func** unExecute()

}

**enum** OperationCode {

**case** plus

**case** minus

**case** divide

**case** multiply

}

**class** ArithmeticUnit {

**private(set)** **var** register: Double

**init**(register: Double) {

**self**.register = register

}

**func** run(code: OperationCode, operand: Double) {

**switch** code {

**case** .plus:

register += operand

**case** .minus:

register -= operand

**case** .divide:

register /= operand

**case** .multiply:

register \*= operand

}

}

}

**class** ControlUnit {

**var** commands: [Command] = []

**var** current: Int = 0

**func** storeCommand(**\_** command: Command) {

commands.append(command)

}

**func** executeCommand() {

commands[current].execute()

current += 1

}

**func** undo() {

commands[current-1].unExecute()

}

**func** redo() {

commands[current-1].execute()

}

}

**class** Add: Command {

**var** operand: Double

**var** unit: ArithmeticUnit

**init**(operand: Double, unit: ArithmeticUnit) {

**self**.operand = operand

**self**.unit = unit

}

**func** execute() {

unit.run(code: .plus, operand: operand)

}

**func** unExecute() {

unit.run(code: .minus, operand: operand)

}

}

**class** Calculator {

**var** arithmeticUnit: ArithmeticUnit = ArithmeticUnit(register: 0)

**var** controlUnit: ControlUnit = ControlUnit()

**func** run(**\_** command: Command) -> Double {

controlUnit.storeCommand(command)

controlUnit.executeCommand()

**return** arithmeticUnit.register

}

**func** add(**\_** operand: Double) -> Double {

run(Add(operand: operand, unit: arithmeticUnit))

}

**func** undo() {

controlUnit.undo()

}

**func** redo() {

controlUnit.redo()

}

}

**var** calculator = Calculator()

**var** result: Double = 0

result = calculator.add(5)

print(result)

result = calculator.add(4)

print(result)

result = calculator.add(3)

print(result)

calculator.undo()

print(calculator.arithmeticUnit.register)

**Паттерн Декоратор**

**import** UIKit

**protocol** AutoBase {

**var** name: String { **get** }

**var** description: String { **get** }

**var** cost: Double { **get** }

**var** toString: String { **get** }

**func** getCost() -> Double

}

**extension** AutoBase {

**var** toString: String {

"Auto - \(name), description - \(description), cost - \(cost)"

}

}

**class** Renault: AutoBase {

**var** name: String

**var** description: String

**var** cost: Double

**func** getCost() -> Double {

cost \* 1.18

}

**init**(name: String, description: String, cost: Double) {

**self**.name = name

**self**.description = description

**self**.cost = cost

}

}

**class** DecoratorOptions: AutoBase {

**var** name: String {

autoProperty.name

}

**var** description: String {

autoProperty.description

}

**var** cost: Double {

autoProperty.cost

}

**func** getCost() -> Double {

autoProperty.getCost()

}

**let** autoProperty: AutoBase

**var** title: String

**init**(auto: AutoBase, title: String) {

**self**.autoProperty = auto

**self**.title = title

}

}

**class** MediaNAV: DecoratorOptions {

**var** auto: AutoBase

**override** **var** name: String {

autoProperty.name + "Modern"

}

**override** **var** description: String {

autoProperty.description + "updated stereo"

}

**override** **var** cost: Double {

autoProperty.cost + 15

}

**override** **func** getCost() -> Double {

autoProperty.getCost() + 15

}

**init**(auto: AutoBase) {

**self**.auto = auto

**super**.init(auto: auto, title: "Media")

}

}

**class** SystemSecurity: DecoratorOptions {

**var** auto: AutoBase

**override** **var** name: String {

autoProperty.name + "Security System"

}

**override** **var** description: String {

autoProperty.description + "high safety"

}

**override** **var** cost: Double {

autoProperty.cost + 15

}

**override** **func** getCost() -> Double {

autoProperty.getCost() + 15

}

**init**(auto: AutoBase) {

**self**.auto = auto

**super**.init(auto: auto, title: "Security")

}

}

**var** reno = Renault(name: "Renault", description: "Logan Active", cost: 500)

print(reno)

**var** myReno: AutoBase = MediaNAV(auto: reno)

print(myReno)

**var** securedReno: AutoBase = SystemSecurity(auto: myReno)

print(securedReno)

**Паттерн Фасад**

**import** UIKit

**class** EventHandler {

**func** handle(**\_** event: String) {

print(event)

}

}

**class** Drive {

**var** driveEvent: EventHandler

**private** **var** twist: String = "" {

**didSet** {

driveEvent.handle(twist)

}

}

**var** toString: String {

"Привод \(twist)"

}

**init**(driveEvent: EventHandler = EventHandler()) {

**self**.driveEvent = driveEvent

}

**func** drive() {

twist = "Starting"

}

**func** turnLeft() {

twist = "Turning left"

}

**func** turnRight() {

twist = "Turning right"

}

**func** stop() {

twist = "Stop"

}

}

**class** Power {

**var** powerEvent: EventHandler

**var** microwavePower: String = ""{

**didSet** {

powerEvent.handle(microwavePower)

}

}

**var** toString: String {

"Мощность \(microwavePower)"

}

**init**(powerEvent: EventHandler = EventHandler()) {

**self**.powerEvent = powerEvent

}

}

**class** Notification {

**var** notificationEvent: EventHandler

**private** **var** message: String = ""{

**didSet** {

notificationEvent.handle(message)

}

}

**var** toString: String {

"Информация \(message)"

}

**init**(notificationEvent: EventHandler = EventHandler()) {

**self**.notificationEvent = notificationEvent

}

**func** startNotification() {

message = "Operation started"

}

**func** stopNotification() {

message = "Operation finished"

}

}

**class** Microwave {

**private** **let** drive: Drive

**private** **let** power: Power

**private** **let** notification: Notification

**init**(drive: Drive, power: Power, notification: Notification) {

**self**.drive = drive

**self**.power = power

**self**.notification = notification

}

**func** defrost() {

notification.startNotification()

power.microwavePower = "1000"

drive.turnRight()

drive.stop()

}

}

**class** Program {

**func** main() {

**let** drive = Drive()

**let** power = Power()

**let** notification = Notification()

**let** microwave = Microwave(drive: drive, power: power, notification: notification)

print("Starting defrost")

microwave.defrost()

}

}

Program().main()

**Паттерн Фабричный метод**

**import** UIKit

**protocol** TransportService {

**var** name: String { **get** }

**func** transportationCost(distance: Double) -> Double

}

**protocol** TransportCompany {

**var** name: String { **get** }

**var** toString: String { **get** }

**func** create(name: String, category: Int) -> TransportService

}

**class** TaxiService: TransportService {

**var** name: String

**var** category: Int

**init**(name: String, category: Int) {

**self**.name = name

**self**.category = category

}

**func** transportationCost(distance: Double) -> Double { distance \* Double(category) }

**var** toString: String { "\(name) - категория \(category)"}

}

**class** ShippingService: TransportService {

**var** name: String

**var** plan: Int

**init**(name: String, plan: Int) {

**self**.name = name

**self**.plan = plan

}

**func** transportationCost(distance: Double) -> Double { distance \* Double(plan) }

**var** toString: String { "\(name) - тариф \(plan)"}

}

**class** TaxiCompany: TransportCompany {

**var** name: String

**var** toString: String { name }

**init**(name: String) {

**self**.name = name

}

**func** create(name: String, category: Int) -> TransportService {

TaxiService(name: name, category: category)

}

}

**class** ShippingCompany: TransportCompany {

**var** name: String

**var** toString: String { name }

**init**(name: String) {

**self**.name = name

}

**func** create(name: String, category: Int) -> TransportService {

ShippingService(name: name, plan: category)

}

}

**let** transportCompany: TransportCompany = TaxiCompany(name: "Taxi Service")

**let** transportService: TransportService = transportCompany.create(name: "Taxi", category: 1)

**var** dist: Double = 15.5

**func** \_print(service: TransportService, dist: Double) {

print("Компания \(service.name), расстоние \(dist) стоимость \(service.transportationCost(distance: dist))")

}

\_print(service: transportService, dist: dist)

**let** gCom: TransportCompany = ShippingCompany(name: "Shipping Service")

**let** gService: TransportService = gCom.create(name: "Shipping", category: 2)

**var** dist2: Double = 150.5

\_print(service: gService, dist: dist2)

// MARK: - Задание Видно, создание объектов нового типа осуществляется довольно легко необходимо лишь, чтобы они поддерживали нужные интерфейсы

**class** DrunkDriverCompany: TransportCompany {

**var** name: String

**var** toString: String { name }

**init**(name: String) {

**self**.name = name

}

**func** create(name: String, category: Int) -> TransportService {

DrunkDriverService(name: name, plan: category)

}

}

**class** DrunkDriverService: TransportService {

**var** name: String

**var** plan: Int

**init**(name: String, plan: Int) {

**self**.name = name

**self**.plan = plan

}

**func** transportationCost(distance: Double) -> Double { distance \* Double(plan) }

**var** toString: String { "\(name) - тариф \(plan)"}

}

**let** dCom: TransportCompany = DrunkDriverCompany(name: "Drunk driver Service")

**let** dService: TransportService = dCom.create(name: "Drunk Driver", category: 3)

**var** dist3: Double = 150.5

\_print(service: dService, dist: dist3)

**Паттерн Синглтон**

**import** UIKit

**class** LogWriter {

**func** log(message: String) {

print("Log Entry:")

print(Date(), "Action:", message)

}

}

**class** Log {

**private** **init** () {}

**static** **let** shared = Log()

**func** logExecution(**\_** message: String) {

**Self**.Loger(logMessage: message, writer: LogWriter())

}

**private** **static** **func** Loger(logMessage: String, writer: LogWriter) {

writer.log(message: logMessage)

}

}

**enum** MathOp {

**case** plus

**case** minus

**case** multiply

**case** divide

}

**class** Operation {

**static** **func** run(operationCode: MathOp, operand: Double) -> Double{

**let** lg2 = Log.shared

**var** rez: Double = 0

**switch** operationCode {

**case** .plus:

lg2.logExecution("Adding + \(operand)")

rez += operand

**case** .minus:

lg2.logExecution("Minus + \(operand)")

rez -= operand

**case** .multiply:

lg2.logExecution("Multipply \* \(operand)")

rez \*= operand

**case** .divide:

lg2.logExecution("Divide / \(operand)")

rez /= operand

}

**return** rez

}

}

**var** lg = Log.shared

lg.logExecution("Start")

**var** op = Operation.run(operationCode: .minus, operand: 35)

op = Operation.run(operationCode: .plus, operand: 30)

**Паттерн Стратегия**

**import** UIKit

**protocol** StrategySort {

**var** title: String { **get** **set** }

**func** sort(array: **inout** [Int]) -> Void

}

**class** InsertionSort: StrategySort {

**var** title: String = "Insertion Sort"

**func** sort(array: **inout** [Int]) {

**guard** array.count >= 2 **else** {

**return**

}

**for** current **in** 1..<array.count {

**for** shifting **in** (1...current).reversed() {

**if** array[shifting] < array[shifting - 1] {

array.swapAt(shifting, shifting - 1)

} **else** {

**break**

}

}

}

}

}

**class** SelectionSort: StrategySort {

**var** title: String = "Strategy Sort"

**func** sort(array: **inout** [Int]) {

**for** current **in** 0..<(array.count - 1) {

**var** lowest = current

**for** other **in** (current + 1)..<array.count {

**if** array[lowest] > array[other] {

lowest = other

}

}

**if** lowest != current {

array.swapAt(lowest, current)

}

}

}

}

**class** Context {

**private** **let** strategy: StrategySort

**private** **var** array: [Int]

**init**(array: [Int], strategy: StrategySort) {

**self**.array = array

**self**.strategy = strategy

}

**func** sort() {

strategy.sort(array: &array)

}

}

**var** arr1 = [31, 15, 20, 2, 4, 10, 68]

**let** sort = SelectionSort()

**let** context = Context(array: arr1, strategy: sort)

context.sort()

print(arr1)

**var** arr2 = [1, 2, 10, 6, 4]

**let** sort2 = InsertionSort()

**let** context2 = Context(array: arr2, strategy: sort2)

context.sort()

print(arr2)

**class** BubbleSort: StrategySort {

**var** title: String = "Bubble sort"

**func** sort(array: **inout** [Int]) {

**guard** array.count >= 2 **else** {

**return**

}

**for** end **in** (1..<array.count).reversed() {

**var** swapped = **false**

**for** current **in** 0..<end {

**if** array[current] > array[current + 1] {

array.swapAt(current, current + 1)

swapped = **true**

}

}

**if** !swapped {

**return**

}

}

}

}

**Паттерн Шаблонный метод**

**import** UIKit

**protocol** Progression {

**var** first: Int { **get** **set** }

**var** last: Int { **get** **set** }

**var** step: Int { **get** **set** }

**var** list: [Int] { **get** **set** }

**func** progress()

}

**extension** Progression {

**mutating** **func** templateMethod() {

initializeProgression(first: first, last: last, step: step)

progress()

print(list)

}

**mutating** **func** initializeProgression(first: Int, last: Int, step: Int) {

**self**.first = first

**self**.last = last

**self**.step = step

}

}

**class** ArithmeticProgression: Progression {

**var** first: Int

**var** last: Int

**var** step: Int

**var** list: [Int] = []

**init**(first: Int, last: Int, step: Int) {

**self**.first = first

**self**.last = last

**self**.step = step

}

**func** progress() {

**var** firstElement = first

**while** firstElement < last {

list.append(firstElement)

firstElement += step

}

}

}

**var** progression: Progression = ArithmeticProgression(first: 1, last: 20, step: 3)

progression.templateMethod()