Практическая работа № 8

Методы

Цель работы:

1. Изучить работу методов, научиться применять на практике.
2. Получить навыки использования методов.
3. Создать консольное приложение, работа которого будет построена на использовании методов.

Краткие теоретические сведения

В языке Swift функции, связанные с определенными типами, называются методами. В Objective C классы используются для определения методов, в то время как гибкийязык Swift предоставляет пользователю возможность иметь методы для классов, структур и перечислений.

## Методы экземпляра

В языке Swiftдоступ к экземплярам Classes, Structures и Enumeration осуществляется через методы экземпляра.

Методы экземпляра обеспечивают функциональность

* Чтобы получить доступ и изменить свойства экземпляра
* функциональность, связанная с потребностью экземпляра

Метод экземпляра может быть записан в фигурных скобках {}. Он имеет неявный доступ к методам и свойствам экземпляра типа. Когда вызывается конкретный экземпляр типа, он получает доступ к этому конкретному экземпляру.

### Синтаксис

funcfuncname(Parameters) ->returntype {

Statement1

Statement2

---

Statement N

return parameters

}

*class calculations {*

*let a:Int*

*let b:Int*

*let res:Int*

*init(a:Int, b:Int){*

*self.a= a*

*self.b= b*

*res= a + b*

*}*

*func tot(c:Int)->Int{*

*return res - c*

*}*

*func result(){*

*print("Result is: \(tot(c: 20))")*

*print("Result is: \(tot(c: 50))")*

*}*

*}*

*letpri= calculations(a:600, b:300)*

*pri.result()*

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

*Resultis: 880*

*Resultis: 850*

Вычисления класса определяют два метода экземпляра –

* init () определен, чтобы сложить два числа a и b и сохранить их в результате ‘res’
* tot () используется, чтобы вычесть ‘res’ из переданного значения ‘c’

Наконец, для печати расчетов используются методы со значениями для a и b. Методы экземпляра доступны с помощью «.» точечный синтаксис

## Локальные и внешние имена параметров

Функции Swift описывают как локальные, так и глобальные объявления для своих переменных. Точно так же соглашения об именах Swift методы также напоминают Objective C. Но характеристики локальных и глобальных объявлений имен параметров различаются для функций и методов. Первый параметр в Swift именуется предлогами как «with», «for» и «by» для простого доступа к соглашениям об именах.

Swift обеспечивает гибкость методов, объявляя имя первого параметра в качестве локальных имен параметров, а оставшиеся имена параметров – как глобальные именапараметров. Здесь ‘no1’ объявляется методами Swift как локальные имена параметров. «no2» используется для глобальных объявлений и доступен через программу.

*class division {*

*var count:Int=0*

*funcincrementBy(no1:Int, no2:Int){*

*count= no1 / no2*

*print(count)*

*}*

*}*

*let counter = division()*

*counter.incrementBy(no1:1800, no2:3)*

*counter.incrementBy(no1:1600, no2:5)*

*counter.incrementBy(no1:11000, no2:3)*

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

*600*

*320*

*3666*

## Имя внешнего параметра с символом # и \_

Несмотря на то, что методы Swift предоставляют первые имена параметров для локальных объявлений, пользователь может изменить имена параметров с локальных на глобальные объявления. Это можно сделать, добавив префикс «#» к имени первого параметра. Таким образом, к первому параметру можно получить глобальный доступ по всем модулям.

Когда пользователю необходимо получить доступ к последующим именам параметров с помощью внешнего имени, имя метода переопределяется с помощью символа «\_».

*class multiplication {*

*var count:Int=0*

*funcincrementBy(no1:Int, no2:Int){*

*count= no1 \* no2*

*print(count)*

*}*

*}*

*let counter = multiplication()*

*counter.incrementBy(no1:800, no2:3)*

*counter.incrementBy(no1:100, no2:5)*

*counter.incrementBy(no1:15000, no2:3)*

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

*2400*

*500*

*45000*

## Ссылка на текущий объект в методах

Методы имеют неявное свойство, известное как «self», для всех экземпляров определенного типа. Свойство «Self» используется для ссылки на текущие экземпляры для его определенных методов.

*class calculations {*

*let a:Int*

*let b:Int*

*let res:Int*

*init(a:Int, b:Int){*

*self.a= a*

*self.b= b*

*res= a + b*

*print("Inside Self Block: \(res)")*

*}*

*func tot(c:Int)->Int{*

*return res - c*

*}*

*func result(){*

*print("Result is: \(tot(c: 20))")*

*print("Result is: \(tot(c: 50))")*

*}*

*}*

*Let pri= calculations(a:600, b:300)*

*let sum = calculations(a:1200, b:300)*

*pri.result()*

*sum.result()*

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

*Inside Self Block: 900*

*Inside Self Block: 1500*

*Result is: 880*

*Result is: 850*

*Result is: 1480*

*Resultis: 1450*

## Изменение типов значений из методов экземпляра

В Swift языковые структуры и перечисления относятся к типам значений, которые не могут быть изменены его методами экземпляра. Тем не менее, язык Swift обеспечивает гибкость для изменения типов значений путем «мутации» поведения. Mutate внесет любые изменения в методы экземпляра и вернется к исходной форме после выполнения метода. Также с помощью свойства «self» создается новый экземпляр для его неявной функции, который заменит существующий метод после его выполнения.

*struct area {*

*var length =1*

*var breadth =1*

*func area()->Int{*

*return length \* breadth*

*}*

*mutatingfuncscaleBy(res:Int){*

*length\*= res*

*breadth\*= res*

*print(length)*

*print(breadth)*

*}*

*}*

*varval= area(length:3, breadth:5)*

*val.scaleBy(res:3)*

*val.scaleBy(res:30)*

*val.scaleBy(res:300)*

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

*9*

*15*

*270*

*450*

*81000*

*135000*

## Ссылка на текущий объект для метода мутации

Методы мутации в сочетании со свойством self присваивают новый экземпляр определенному методу.

*struct area {*

*var length =1*

*var breadth =1*

*func area()->Int{*

*return length \* breadth*

*}*

*mutatingfuncscaleBy(res:Int){*

*self.length\*= res*

*self.breadth\*= res*

*print(length)*

*print(breadth)*

*}*

*}*

*var val= area(length:3, breadth:5)*

*val.scaleBy(res:13)*

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат. –

*39*

*65*

Задание для самостоятельного выполнения

1. В соответствии со своим вариантом задания создать структуру, инициализировать его, реализовать методы мутации для численного значения, которое может быть увеличено, без потери здравого смысла объекта.
2. Реализовать метод вывода на печать экземпляра структуры
3. Создать массив из 5 элементов типа созданной структуры согласно Вашему варианту задания. (Автопарк, библиотека, каталог, команда, группа, реестр фирм, склад компьютеров, спортклуб, автостоянка, каталог журнальных статей).
4. Реализовать метод поиска нужного объекта в массиве и вывести его на экран

Варианты

1. Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту)
2. Книга (название книги, автор, издательство, количество страниц)
3. Файл (имя, время создания, дата создания, размер)
4. Спортсмен (ФИО, вид спорта, сколько лет занимается)
5. Студент (ФИО, группа, средний балл зачетки)
6. Фирма (название, вид услуг, годовой оборот)
7. Компьютер (производитель, частота процессора, количество ядер, объем памяти)
8. Спорт (вид спорта, количество занимающихся в городе, средняя стоимость комплекта инвентаря)
9. Автомобиль (марка, фирма производитель, год выпуска, пробег)
10. Статья (название, автор, название журнала, год и месяц издания)