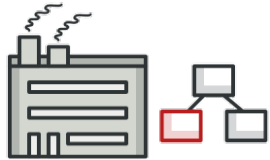





[🏠](#) / [Паттерны проектирования](#) / [Фабричный метод](#) / [Go](#)



Фабричный метод на Go

Фабричный метод — это порождающий паттерн проектирования, который решает проблему создания различных продуктов, без указания конкретных классов продуктов.

Фабричный метод задаёт метод, который следует использовать вместо вызова оператора `new` для создания объектов-продуктов. Подклассы могут переопределить этот метод, чтобы изменять тип создаваемых продуктов.

 [Подробнее о паттерне Фабричный метод →](#)

Навигация

 [Интро](#)

 [Концептуальный пример](#)


 [iGun](#)

[Главная](#) [Рефакторинг](#) [Паттерны](#) [Премиум контент](#)
[Форум](#) [Связаться](#)



© 2014-2023 Refactoring.Guru.

Все права защищены.

 Иллюстрации нарисовал Дмитрий Жарт

[Условия использования](#)

[Политика конфиденциальности](#)

[Использование контента](#) [About us](#)

Концептуальный пример

В Go невозможно реализовать классический вариант паттерна Фабричный метод, поскольку в языке отсутствуют возможности ООП, в том числе классы и наследственность. Несмотря на это, мы все же можем реализовать базовую версию этого паттерна — Простая фабрика.

В этом примере мы будем создавать разные типы оружия при помощи структуры фабрики.

Сперва, мы создадим интерфейс `iGun`, который определяет все методы будущих пушек. Также имеем структуру `gun` (пушка), которая применяет интерфейс `iGun`. Две конкретных пушки — `ak47` и `musket` — обе включают в себя структуру `gun` и не напрямую реализуют все методы от `iGun`.

`gunFactory` служит фабрикой, которая создает пушку нужного типа в зависимости от аргумента на входе. Клиентом служит `main.go`. Вместо прямого взаимодействия с объектами `ak47` или `musket`, он создает экземпляры различного оружия при помощи `gunFactory`, используя для контроля изготовления только параметры в виде строк.

iGun.go: Интерфейс продукта

```
package main

type IGun interface {
    setName(name string)
    setPower(power int)
    getName() string
    getPower() int
}
```

gun.go: Конкретный продукт

```
package main

type Gun struct {
    name string
    power int
}
```

```

func (g *Gun) setName(name string) {
    g.name = name
}

func (g *Gun) getName() string {
    return g.name
}

func (g *Gun) setPower(power int) {
    g.power = power
}

func (g *Gun) getPower() int {
    return g.power
}

```

ak47.go: Конкретный продукт

```

package main

type Ak47 struct {
    Gun
}

func newAk47() IGun {
    return &Ak47{
        Gun: Gun{
            name: "AK47 gun",
            power: 4,
        },
    }
}

```

musket.go: Конкретный продукт

```

package main

type musket struct {
    Gun
}

```

```

func newMusket() IGun {
    return &musket{
        Gun: Gun{
            name: "Musket gun",
            power: 1,
        },
    }
}

```

gunFactory.go: Фабрика

```

package main

import "fmt"

func getGun(gunType string) (IGun, error) {
    if gunType == "ak47" {
        return newAk47(), nil
    }
    if gunType == "musket" {
        return newMusket(), nil
    }
    return nil, fmt.Errorf("Wrong gun type passed")
}

```

main.go: Клиентский код

```

package main

import "fmt"

func main() {
    ak47, _ := getGun("ak47")
    musket, _ := getGun("musket")

    printDetails(ak47)
    printDetails(musket)
}

func printDetails(g IGun) {
    fmt.Printf("Gun: %s", g.getName())
}

```

```
fmt.Println()  
fmt.Printf("Power: %d", g.getPower())  
fmt.Println()  
}
```

output.txt: Результат выполнения

```
Gun: AK47 gun  
Power: 4  
Gun: Musket gun  
Power: 1
```

По материалам: *Golang By Example*

ВЕРНУТЬСЯ НАЗАД

ЧИТАЕМ ДАЛЬШЕ

← Строитель на Go

Прототип на Go →

Фабричный метод на других языках программирования

