DWARVES FOUNDATION

Strategy

One of the behavior patterns

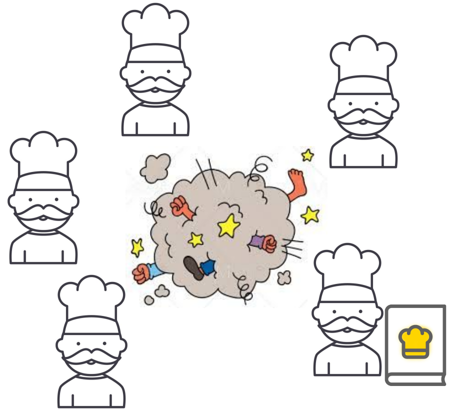
Problem statement

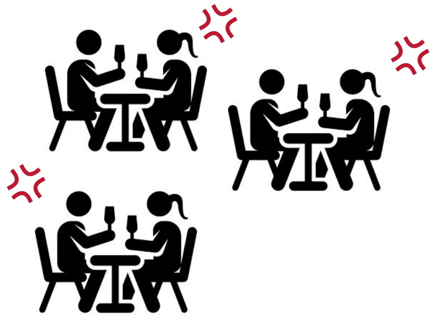
- A prestigious restaurant has a secret cookbook. The book contains many recipes from many cuisine(asian, europe, etc…). Only the head cook are allowed to view/change the cookbook.

DWARVES FOUNDATION

Problem statement

- To keep up with the growth, the restaurant hires more cooks and head cooks but still want to keep a single secret cookbook. As result, head cooks fighting with each others to take view/edit priority during rush hours. The restaurant can not copy the cookbook to avoid leaking important trade secret



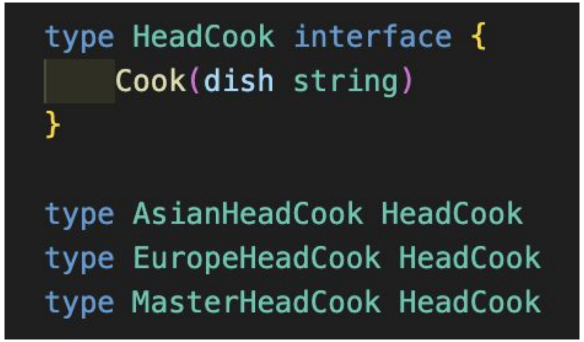


Problem statement

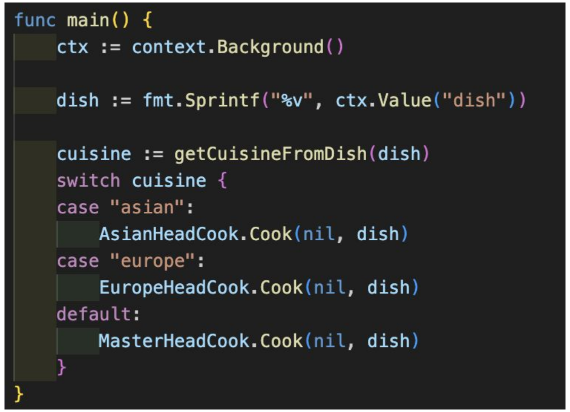
- We devise a strategy to divide the cookbook into multiple parts. Each part responsible for a cultural cuisine. The head cooks with specific cuisine specialty can only view/edit their respective parts.

Problem statement

- We devise a strategy to divide the cookbook into multiple parts. Each part responsible for a cultural cuisine. The head cooks with specific cuisine specialty can only view/edit their respective parts.



```go
type HeadCook interface {
	Cook(dish string)
}

type AsianHeadCook HeadCook
type EuropeHeadCook HeadCook
type MasterHeadCook HeadCook
```



```go
func main() {
	ctx := context.Background()

	dish := fmt.Sprintf("%v", ctx.Value("dish"))

	cuisine := getCuisineFromDish(dish)
	switch cuisine {
	case "asian":
		AsianHeadCook.Cook(nil, dish)
	case "europe":
		EuropeHeadCook.Cook(nil, dish)
	default:
		MasterHeadCook.Cook(nil, dish)
	}
}
```

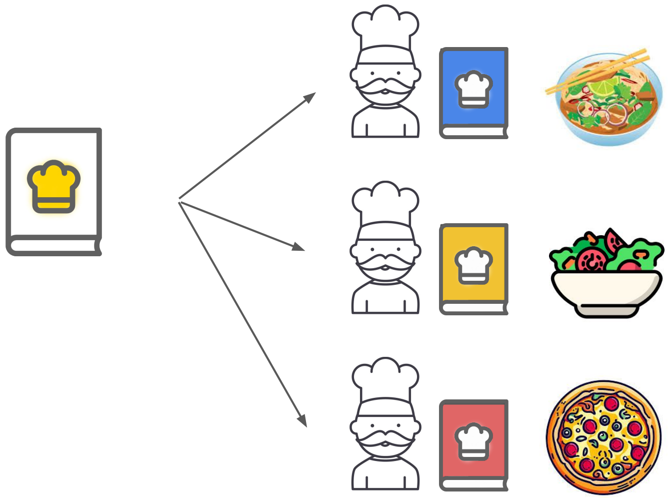
DWARVES FOUNDATION

Strategy

- Strategy is dividing a class that does something specific in a lot of different ways into strategies, each strategy is created as a solution to the expected provided context.

Strategy

- Strategy divides the cookbook(class) which is followed to cook food(something specific) into parts based on geological cuisines(different ways).



DWARVES FOUNDATION

Strategy

- We can divide a class into strategies, by levels of abstraction and The behaviours for specific contexts.

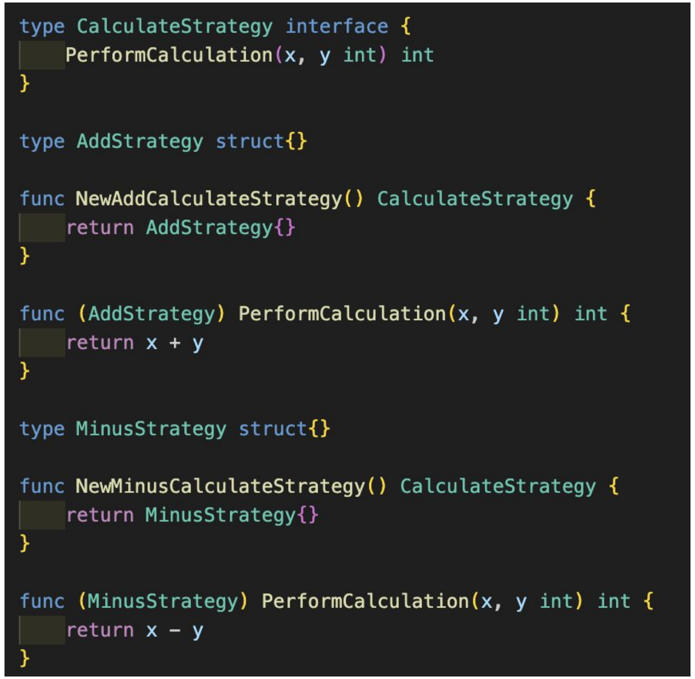
Strategy

- FindWay(from, to, vehicle).
    - FindWayLand(from, to, vehicle).
        - FindWayCar(from, to, vehicle).
        - FindWayBus(from, to, vehicle).
    - FindWayAir(from, to, vehicle).
    - FindWaySea(from, to, vehicle).

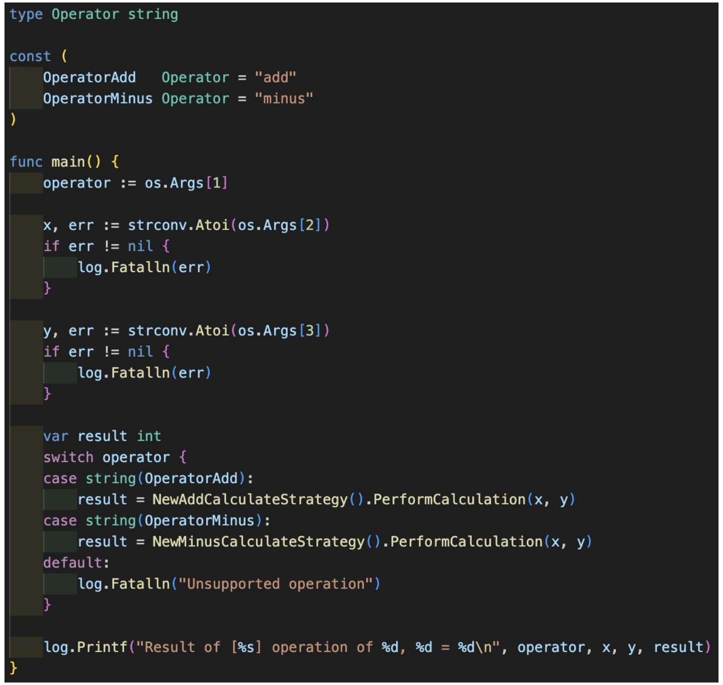
Strategy

After defining the strategies, we have the client select the appropriate strategy based on the provided context

Go example



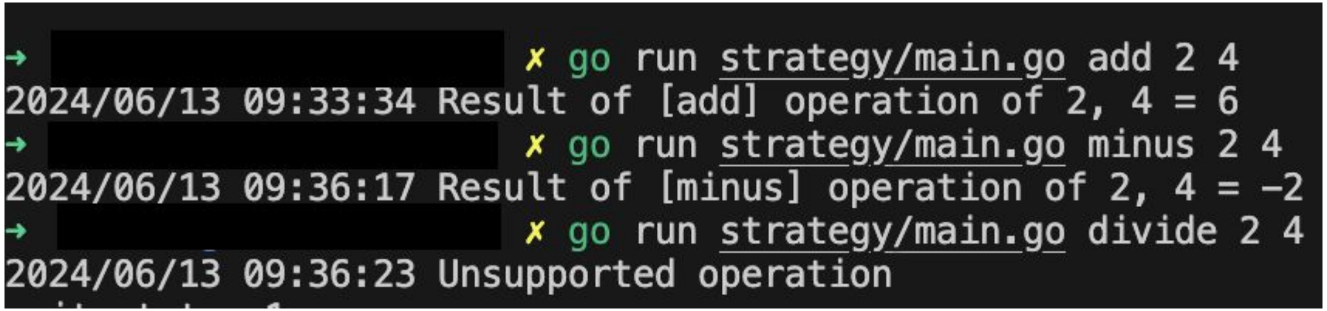
```go
type CalculateStrategy interface {
	PerformCalculation(x, y int) int
}

type AddStrategy struct{}

func NewAddCalculateStrategy() CalculateStrategy {
	return AddStrategy{}
}

func (AddStrategy) PerformCalculation(x, y int) int {
	return x + y
}

type MinusStrategy struct{}

func NewMinusCalculateStrategy() CalculateStrategy {
	return MinusStrategy{}
}

func (MinusStrategy) PerformCalculation(x, y int) int {
	return x - y
}
```



```go
type Operator string

const (
	OperatorAdd   Operator = "add"
	OperatorMinus Operator = "minus"
)

func main() {
	operator := os.Args[1]

	x, err := strconv.Atoi(os.Args[2])
	if err != nil {
		log.Fatalln(err)
	}

	y, err := strconv.Atoi(os.Args[3])
	if err != nil {
		log.Fatalln(err)
	}

	var result int
	switch operator {
	case string(OperatorAdd):
		result = NewAddCalculateStrategy().PerformCalculation(x, y)
	case string(OperatorMinus):
		result = NewMinusCalculateStrategy().PerformCalculation(x, y)
	default:
		log.Fatalln("Unsupported operation")
	}

	log.Printf("Result of [%s] operation of %d, %d = %d\n", operator, x, y, result)
}
```

DWARVES FOUNDATION

Go example



```
→ [redacted] ✗ go run strategy/main.go add 2 4
2024/06/13 09:33:34 Result of [add] operation of 2, 4 = 6
→ [redacted] ✗ go run strategy/main.go minus 2 4
2024/06/13 09:36:17 Result of [minus] operation of 2, 4 = -2
→ [redacted] ✗ go run strategy/main.go divide 2 4
2024/06/13 09:36:23 Unsupported operation
```

DWARVES FOUNDATION

Why use strategy

Strategy patterns give each the abilities to:

- Create and cherry-pick between strategies.
- Each strategy is independent of each other.
- Client select the strategy so can isolate the implementation details of a strategy from the code that uses it

Notes

Before using the strategy pattern:

- To not overcomplicate the program if the context are rarely change and the strategies are few and simple.
- The client must aware the differences between strategies to be able to select a proper one.

DWARVES
FOUNDATION

Q&A

References

- https://refactoring.guru/design-patterns/strategy.