

Если вы видите что-то необычное, просто сообщите мне.

# Спецификация (Specification)

Спецификация — шаблон проектирования, посредством которого представление правил бизнес логики может быть преобразовано в виде цепочки объектов, связанных операциями булевой логики.

Больше информации в Wikipedia [https://en.wikipedia.org/wiki/Specification\\_pattern](https://en.wikipedia.org/wiki/Specification_pattern)

```
//specification.go
// Pattern Specification
//
// In the following example, we are retrieving invoices and sending them to a collection agency if
// 1. they are overdue,
// 2. notices have been sent, and
// 3. they are not already with the collection agency.
// This example is meant to show the end result of how the logic is 'chained' together.
//
// This usage example assumes a previously defined OverdueSpecification class
// that is satisfied when an invoice's due date is 30 days or older,
// a NoticeSentSpecification class that is satisfied when three notices
// have been sent to the customer, and an InCollectionSpecification class
// that is satisfied when an invoice has already been sent to the collection
// agency. The implementation of these classes isn't important here.

package specification

// Data for analysis
type Invoice struct {
    Day    int
    Notice int
    IsSent bool
```

```
}
```

```
////
```

```
// Invoice Specification Interface
```

```
type Specification interface {
```

```
  IsSatisfiedBy(Invoice) bool
```

```
  And(Specification) Specification
```

```
  Or(Specification) Specification
```

```
  Not() Specification
```

```
  Relate(Specification)
```

```
}
```

```
////
```

```
// Invoice BaseSpecification
```

```
type BaseSpecification struct {
```

```
  Specification
```

```
}
```

```
// Check specification
```

```
func (self *BaseSpecification) IsSatisfiedBy(elm Invoice) bool {
```

```
  return false
```

```
}
```

```
// Condition AND
```

```
func (self *BaseSpecification) And(spec Specification) Specification {
```

```
  a := &AndSpecification{
```

```
    self.Specification, spec,
```

```
  }
```

```
  a.Relate(a)
```

```
  return a
```

```
}
```

```
// Condition OR
```

```
func (self *BaseSpecification) Or(spec Specification) Specification {
```

```
  a := &OrSpecification{
```

```
    self.Specification, spec,
```

```
  }
```

```
  a.Relate(a)
```

```

    return a
}

// Condition NOT
func (self *BaseSpecification) Not() Specification {
    a := &NotSpecification{
        self.Specification,
    }
    a.Relate(a)
    return a
}

// Relate to specification
func (self *BaseSpecification) Relate(spec Specification) {
    self.Specification = spec
}

/////

// AndSpecification
type AndSpecification struct {
    Specification
    compare Specification
}

// Check specification
func (self *AndSpecification) IsSatisfiedBy(elm Invoice) bool {
    return self.Specification.IsSatisfiedBy(elm) && self.compare.IsSatisfiedBy(elm)
}

/////

// OrSpecification
type OrSpecification struct {
    Specification
    compare Specification
}

// Check specification
func (self *OrSpecification) IsSatisfiedBy(elm Invoice) bool {

```

```

    return self.Specification.IsSatisfiedBy(elm) || self.compare.IsSatisfiedBy(elm)
}

////

// NotSpecification
type NotSpecification struct {
    Specification
}

// Check specification
func (self *NotSpecification) IsSatisfiedBy(elm Invoice) bool {
    return !self.Specification.IsSatisfiedBy(elm)
}

////

// Invoice's due date is 30 days or older
type OverDueSpecification struct {
    Specification
}

// Check specification
func (self *OverDueSpecification) IsSatisfiedBy(elm Invoice) bool {
    return elm.Day >= 30
}

// Constructor
func NewOverDueSpecification() Specification {
    a := &OverDueSpecification{&BaseSpecification{}}
    a.Relate(a)
    return a
}

// Three notices have been sent to the customer
type NoticeSentSpecification struct {
    Specification
}

// Check specification

```

```

func (self *NoticeSentSpecification) IsSatisfiedBy(elm Invoice) bool {
    return elm.Notice >= 3
}

// Constructor
func NewNoticeSentSpecification() Specification {
    a := &NoticeSentSpecification{&BaseSpecification{}}
    a.Relate(a)
    return a
}

// Invoice has already been sent to the collection agency.
type InCollectionSpecification struct {
    Specification
}

// Check specification
func (self *InCollectionSpecification) IsSatisfiedBy(elm Invoice) bool {
    return !elm.IsSent
}

// Constructor
func NewInCollectionSpecification() Specification {
    a := &InCollectionSpecification{&BaseSpecification{}}
    a.Relate(a)
    return a
}

```

```

//specification_test.go
package specification

import (
    "testing"
)

func TestSpecification(t *testing.T) {
    overDue := NewOverDueSpecification()
    noticeSent := NewNoticeSentSpecification()
    inCollection := NewInCollectionSpecification()

    sendToCollection := overDue.And(noticeSent).And(inCollection.Not())
}

```

```
invoice := Invoice{
  Day: 31, // >= 30
  Notice: 4, // >= 3
  IsSent: false, // false
}

// true!
result := sendToCollection.IsSatisfiedBy(invoice)

if !result {
  t.Errorf("Expect result to equal %v, but %v.\n", false, true)
}
}
```

---

Revision #1

Created 3 July 2022 10:34:22 by gasick

Updated 3 July 2022 10:35:13 by gasick