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

Наблюдатель (Observer)

Паттерн Observer относится к поведенческим паттернам уровня объекта.

Паттерн Observer определяет зависимость "один-ко-многим" между объектами так, что при изменении состояния одного объекта все зависящие от него объекты уведомляются об этом и обновляются автоматически.

Основные участиники паттерна это Издатели (Subject) и Подписчики (Observer).

Имеется два способа получения уведомлений от издателя:

1. Метод вытягивания: После получения уведомления от издателя, подписчик должен пойти к издателю и забрать (вытянуть) данные самостоятельно.
2. Метод проталкивания: Издатель не уведомляет подписчика об обновлениях данных, а самостоятельно доставляет (проталкивает) данные подписчику.

Требуется для реализации:

1. Абстрактный класс Subject, определяющий интерфейс Издателя;
2. Класс ConcreteSubject, реализует интерфейс Subject;
3. Абстрактный класс Observer, определяющий общий функционал Подписчиков;
4. Класс ConcreteObserver, реализует Подписчика;

[!] В описании паттерна применяются общие понятия, такие как Класс, Объект, Абстрактный класс. Применимо к языку Go, это Пользовательский Тип, Значение этого Типа и Интерфейс. Также в языке Go за место общепринятого наследования используется агрегирование и встраивание.

```
//observer.go
// Package observer is an example of the Observer Pattern.
// Push model.
package observer
```

```
// Publisher interface.
type Publisher interface {
	Attach(observer Observer)
	SetState(state string)
	Notify()
}

// Observer provides a subscriber interface.
type Observer interface {
	Update(state string)
}

// ConcretePublisher implements the Publisher interface.
type ConcretePublisher struct {
	observers []Observer
	state     string
}

// NewPublisher is the Publisher constructor.
func NewPublisher() Publisher {
	return &ConcretePublisher{}
}

// Attach a Observer
func (s *ConcretePublisher) Attach(observer Observer) {
	s.observers = append(s.observers, observer)
}

// SetState sets new state
func (s *ConcretePublisher) SetState(state string) {
	s.state = state
}

// Notify sends notifications to subscribers.
// Push model.
func (s *ConcretePublisher) Notify() {
	for _, observer := range s.observers {
		observer.Update(s.state)
	}
```

```
}

// ConcreteObserver implements the Observer interface.
type ConcreteObserver struct {
	state string
}

// Update set new state
func (s *ConcreteObserver) Update(state string) {
	s.state = state
}
```

```
//observer_test.go
package observer

func ExampleObserver() {

	publisher := NewPublisher()

	publisher.Attach(&ConcreteObserver{})
	publisher.Attach(&ConcreteObserver{})
	publisher.Attach(&ConcreteObserver{})

	publisher.SetState("New State...")

	publisher.Notify()
}
```

Revision #1
Created 3 July 2022 10:16:16 by gasick
Updated 3 July 2022 10:16:58 by gasick