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

?????????? ??????? (Abstract Factory)

Паттерн Abstract Factory относится к порождающим паттернам уровня объекта.

Паттерн Abstract Factory предоставляет общий интерфейс для создания семейства взаимосвязанных объектов. Это позволяет отделить функциональность системы от внутренней реализации каждого класса, а обращение к этим классам становится возможным через абстрактные интерфейсы.

В общем виде абстрактная фабрика выглядит следующим образом. Для каждого из семейств объектов, создается конкретная фабрика (наследник абстрактной), посредством которой создаются продукты этого семейства.

Пример: Есть две фабрики по производству газировки, Кока-Кола и Пепси. Эти фабрики выпускают семейство продуктов (объектов) - бутылка, крышка, этикетка, жидкость. Каждая из этих фабрик выпускает продукты, которые взаимодействуют между собой и не могут жить отдельно друг от друга. Фабрика Кока-Кола не может поставлять клиентам пустые бутылки.

Что бы реализовать простое создание семейства объектов, должен быть интерфейс, по которому работает фабрика, так же фабрика должна выпускать продукты с определенным интерфейсом. Например, бутылки обеих компаний обладают одним интерфейсом - у них есть горлышко через которое они наполняются жидкостью, так же мы можем узнать объем бутылок. Дальше бутылки могут отличаться по форме, объему или материалу, нас это не касается, нам нужно только знать, куда наливать жидкость, а так же, сколько этой жидкости нужно.

Требуется для реализации:

1. Класс абстрактной фабрики AbstractFactory, описывающий общий интерфейс фабрики, от которой будет наследоваться каждая конкретная фабрика;
2. Класс абстрактного продукта AbstractProduct, описывающий общий интерфейс продукта, от которого будет наследоваться каждый конкретный продукт;
3. Класс конкретной фабрики Factory;
4. Класс конкретного продукта ProductA.
5. Класс конкретного продукта ProductB.

Подведем итог.

Абстрактная фабрика представляет собой базовый класс, описывающий интерфейс конкретных фабрик, создающих продукты. Производные от него классы конкретных фабрик, должны реализовать этот интерфейс.

Также абстрактная фабрика должна описывать абстрактные продукты, которые она производит, что бы конкретные фабрики производили продукты с нужными интерфейсами.

[!] 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.

```
//abstract_factory.go
// Package abstract_factory is an example of the Abstract Factory Pattern.
package abstract_factory

// AbstractFactory provides an interface for creating families of related objects.
type AbstractFactory interface {
	CreateWater(volume float64) AbstractWater
	CreateBottle(volume float64) AbstractBottle
}

// AbstractWater provides a water interface.
type AbstractWater interface {
	GetVolume() float64
}

// AbstractBottle provides a bottle interface.
```

```
type AbstractBottle interface {
	PourWater(water AbstractWater) // Bottle interacts with a water.
	GetBottleVolume() float64
	GetWaterVolume() float64
}

// CocaColaFactory implements AbstractFactory interface.
type CocaColaFactory struct {
}

// NewCocaColaFactory is the CocaColaFactory constructor.
func NewCocaColaFactory() AbstractFactory {
	return &CocaColaFactory{}
}

// CreateWater implementation.
func (f *CocaColaFactory) CreateWater(volume float64) AbstractWater {
	return &CocaColaWater{volume: volume}
}

// CreateBottle implementation.
func (f *CocaColaFactory) CreateBottle(volume float64) AbstractBottle {
	return &CocaColaBottle{volume: volume}
}

// CocaColaWater implements AbstractWater.
type CocaColaWater struct {
	volume float64 // Volume of drink.
}

// GetVolume returns volume of drink.
func (w *CocaColaWater) GetVolume() float64 {
	return w.volume
}

// CocaColaBottle implements AbstractBottle.
type CocaColaBottle struct {
	water  AbstractWater // Bottle must contain a drink.
	volume float64       // Volume of bottle.
}
```

```
// PourWater pours water into a bottle.
func (b *CocaColaBottle) PourWater(water AbstractWater) {
	b.water = water
}

// GetBottleVolume returns volume of bottle.
func (b *CocaColaBottle) GetBottleVolume() float64 {
	return b.volume
}

// GetWaterVolume returns volume of water.
func (b *CocaColaBottle) GetWaterVolume() float64 {
	return b.water.GetVolume()
}
```

```
//abstract_factory_test.go
package abstract_factory

import (
	"testing"
)

func TestAbstractFactory(t *testing.T) {

	cocacolaFactory := NewCocaColaFactory()

	cocacolaWater := cocacolaFactory.CreateWater(2.5)
	cocacolaBottle := cocacolaFactory.CreateBottle(2.5)

	cocacolaBottle.PourWater(cocacolaWater)

	if cocacolaBottle.GetWaterVolume() != cocacolaBottle.GetBottleVolume() {
		t.Errorf("Expect volume to %.1fL, but %.1fL", cocacolaBottle.GetWaterVolume(),
cocacolaBottle.GetBottleVolume())
	}
}
```

Revision #1

Created 2022-07-03 10:22:35 UTC by gasick

Updated 2022-07-03 10:23:15 UTC by gasick