

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
∞...☼...∞



BÀI TẬP LỚN MÔN LẬP TRÌNH NÂNG CAO

ĐỀ TÀI:

DESIGN PATTERN - FLYWEIGHT

LỚP L01, L02 --- NHÓM ... --- HK222

NGÀY NỘP

Giảng viên hướng dẫn: Lê Đình Thuận

SINH VIÊN THỰC HIỆN

STT	MSSV	Họ	Tên	%Điểm BTL	Điểm BTL	Ghi chú
1	2011856	Nguyễn Thanh	Phúc	100%		L01
2	2013444	Nguyễn Lê	Khanh	100%		L01
3	2010448	Nguyễn Trung	Nghĩa	100%		L02
4	2011365	Nguyễn Hữu	Khang	100%		L02
5	2112462	Trần Huỳnh Khánh	Toàn	100%		L02

Thành phố Hồ Chí Minh – 2023

TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KH&KT MÁY TÍNH

BÁO CÁO KẾT QUẢ LÀM VIỆC NHÓM VÀ BẢNG ĐIỂM BTL

Môn: LẬP TRÌNH NÂNG CAO (MSMH: CO2039)

Nhóm/Lớp: ...L01, L02..... Tên nhóm:HK222.....Năm học2022 - 2023.....

Đề tài:

FLYWEIGHT DESIGN PATTERN

STT	Mã số SV	Họ	Tên	Nhiệm vụ được phân công	% Điểm BTL	Điểm BTL	Ký tên
1	2011856	Nguyễn Thanh	Phúc	Code demo Flyweight Pattern	100%		
2	2013444	Nguyễn Lê	Khanh	Cơ sở Lý thuyết + Soạn báo cáo	100%		
3	2010448	Nguyễn Trung	Nghĩa	Slide thuyết trình	100%		
4	2011365	Nguyễn Hữu	Khang	Code demo Flyweight Pattern	100%		
5	2112462	Trần Huỳnh Khánh	Toàn	Cơ sở Lý thuyết + Soạn báo cáo	100%		

Họ và tên nhóm trưởng:.....Nguyễn Thanh Phúc....., Số ĐT:0792675776.....

Email:phuc.nguyenedison@hcmut.edu.vn.....

Nhận xét của GV:

GIẢNG VIÊN

(Ký và ghi rõ họ, tên)

NHÓM TRƯỞNG

(Ký và ghi rõ họ, tên)

MỤC LỤC

LỜI MỞ ĐẦU.....	2
Chương 1:.....	3
TỔNG QUAN VỀ DESIGN PATTERN.....	3
1.1. Design Pattern là gì ?.....	3
1.2. Những nhầm lẫn về Design Pattern.....	3
1.3. Cấu trúc Design Pattern.....	4
1.4. Phân loại Design Pattern.....	4
Chương 2:.....	6
FLYWEIGHT DESIGN PATTERN.....	6
2.1. Giới thiệu chung.....	6
2.2. Mục đích của Flyweight Design Pattern.....	6
2.3. Kiến trúc của Flyweight Design Pattern.....	6
2.4. Ví dụ minh họa Flyweight Design Pattern.....	8
2.5. Ưu và nhược điểm của Flyweight Design Pattern.....	11
2.6. Các trường hợp sử dụng Flyweight Design Pattern.....	11
TỔNG KẾT.....	13
TÀI LIỆU THAM KHẢO.....	14

LỜI MỞ ĐẦU

Design Pattern là một trong những khái niệm quan trọng trong lập trình hướng đối tượng. Nó cung cấp cho các nhà phát triển một bộ sưu tập các giải pháp phổ biến và được kiểm chứng để giải quyết các vấn đề thường gặp trong thiết kế phần mềm. Sử dụng Design Pattern giúp cho quá trình phát triển phần mềm trở nên dễ dàng hơn và giảm thiểu các lỗi phát sinh trong quá trình phát triển.

Trong báo cáo này, chúng tôi sẽ tìm hiểu về một trong những Design Pattern phổ biến nhất, đó là Flyweight Design Pattern. Flyweight Pattern là một mẫu thiết kế nhằm giảm thiểu sự lãng phí tài nguyên trong các ứng dụng có quá nhiều đối tượng giống nhau. Chúng ta sẽ cùng tìm hiểu về cách Flyweight Pattern hoạt động và cách áp dụng nó trong thực tế. Hy vọng bạn sẽ có được hiểu biết sâu hơn về Design Pattern và Flyweight Pattern và áp dụng chúng một cách hiệu quả trong các dự án phần mềm của mình.

Chương 1:

TỔNG QUAN VỀ DESIGN PATTERN

1.1. Design Pattern là gì ?

Design pattern nắm bắt chuyên môn mô tả một thiết kế kiến trúc cho một vấn đề thiết kế trong một tình huống cụ thể. Nó cũng chứa thông tin về khả năng ứng dụng của nó, sự đánh đổi phải được thực hiện và bất kỳ nhược điểm nào của giải pháp.

Các design pattern cực kỳ hữu ích cho cả những người thiết kế hướng đối tượng mới làm quen và có kinh nghiệm. Đây là bởi vì họ gói gọn kiến thức thiết kế sâu rộng và các giải pháp thiết kế đã được chứng minh cùng với hướng dẫn về cách sử dụng chúng. Việc sử dụng lại các pattern mở ra một cấp độ tái sử dụng thiết kế bổ sung, trong đó các triển khai khác nhau, nhưng vì kiến trúc được đại diện bởi các pattern vẫn được áp dụng.

Do đó, các pattern cho phép các lập trình viên thiết kế chia sẻ kiến thức về thiết kế kiến trúc phần mềm. Họ nắm bắt các cấu trúc tĩnh và động cũng như sự cộng tác của các giải pháp thành công trước đó cho các vấn đề phát sinh khi xây dựng ứng dụng.

1.2. Những nhầm lẫn về Design Pattern

Design Pattern không phải là mã có thể tái sử dụng. Lí do là vì chúng thường không cụ thể hóa code. Việc thực hành các Design Pattern phụ thuộc vào ngôn ngữ lập trình và thậm chí là người thực hiện nó.

Design Pattern và Principle là hai thứ khác nhau.

Design Pattern không phải là một cấu trúc phần mềm. Cấu trúc phần mềm đưa ra mệnh lệnh cái gì sẽ được thực hành và được đặt tại đâu. Trong khi đó, Design Pattern trình bày nên thực hiện cái gì như thế nào.

Design Pattern không phải là những giải pháp sẵn-sàng-đề-code. Chúng giống như những mô tả giải pháp sẽ trông như thế nào. Những gì bạn nên nhớ từ Design Pattern là một vấn đề và giải pháp có mối liên quan mật thiết. Cả hai đều rất quan trọng cho việc học hỏi.

1.3. Cấu trúc Design Pattern

Hầu hết các Design Pattern đều được trình bày một cách thống nhất. Như vậy, mọi người có thể sử dụng chúng trong mọi trường hợp. Dưới đây là những phần thường được trình bày trong một mẫu thiết kế phần mềm:

- Mục tiêu: thường mô tả ngắn gọn vấn đề và cách giải quyết
- Động lực: giải thích kỹ hơn về vấn đề và giải pháp có thể có được từ Design Pattern
- Cấu trúc các lớp: trình bày từng phần của pattern và cách chúng liên kết với nhau
- Ví dụ code: Đưa ra ví dụ code của một trong những ngôn ngữ lập trình phổ biến nhất để người đọc hiểu được ý tưởng cơ bản của pattern.
- Một vài catalog về design pattern còn liệt kê những chi tiết hữu ích khác như tính ứng dụng của một pattern, các bước triển khai và mối quan hệ với những pattern khác.

1.4. Phân loại Design Pattern

Hiện nay, Design Pattern được chia thành 3 nhóm chính: Khởi tạo (Creational Pattern), Cấu trúc (Structural Pattern) và Hành vi (Behavioral Pattern).

1.4.1. Mẫu thiết kế khởi tạo (Creational Pattern)

Nhóm Creational Pattern gồm 5 mẫu: Singleton, Factory Method, Abstract Factory, Builder, Prototype.

Mẫu thiết kế này được dùng để tạo ra đối tượng cho một class (lớp) thích hợp. Class này sẽ là giải pháp cho vấn đề. Chúng đặc biệt hữu ích khi bạn đang tận dụng tính đa hình và cần phải lựa chọn giữa các class khác nhau trong runtime (thời gian chạy) thay vì compile time (thời gian biên dịch).

Mẫu thiết kế khởi tạo hỗ trợ việc tạo ra các đối tượng trong một hệ thống mà không cần nhận dạng loại lớp cụ thể trong code. Vì vậy, ta không cần phải viết những dòng code lớn, phức tạp để tạo bản sao của một đối tượng. Tuy nhiên, số đối tượng được tạo ra sẽ bị hạn chế.

1.4.2. Mẫu thiết kế cấu trúc (Structural Pattern)

Nhóm Structural Pattern gồm 7 mẫu: Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy.

Mẫu thiết kế cấu trúc tạo ra những cấu trúc lớn hơn từ những phần riêng lẻ, thường là của những class khác nhau. Mẫu này rất đa dạng, tùy thuộc vào dạng cấu trúc nào cần được tạo ra, với mục đích gì. Mẫu thiết kế cấu trúc quan tâm đến cách thức các class và đối tượng được cấu trúc để tạo nên những cấu trúc lớn hơn. Chúng sử dụng tính kế thừa để xây dựng các interface (giao diện) hay implementation (triển khai).

1.4.3. Mẫu thiết kế hành vi (Behavioral Pattern)

Nhóm Behavioral Pattern gồm 11 mẫu: Interpreter, Template Method, Chain of, Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Visitor.

Mẫu thiết kế hành vi mô tả tương tác giữa các đối tượng và tập trung vào cách chúng giao tiếp với nhau. Chúng có thể giảm bớt những lưu đồ phức tạp thành liên kết giữa những đối tượng thuộc các class khác nhau. Mẫu hành vi còn được sử dụng để tạo thuật toán cho một class sử dụng. Nói đơn giản, chúng là một thông số có thể điều chỉnh được trong runtime.

Mẫu thiết kế này quan tâm đến các thuật toán và việc phân trách nhiệm giữa các đối tượng. Chúng không chỉ mô tả mẫu các đối tượng hay các class mà còn cả mẫu sự giao tiếp giữa chúng. Chúng khiến chúng ta quan tâm đến cách các đối tượng liên kết với nhau thay vì dòng kiểm soát. Mẫu thiết kế hành vi sử dụng tính kế thừa để phân phối hành vi giữa các class.

Chương 2:

FLYWEIGHT DESIGN PATTERN

2.1. Giới thiệu chung

Flyweight pattern cho phép giảm số lượng lớn các đối tượng tương tự thành một tập hợp nhỏ các đối tượng được chia sẻ. Điều này có lợi về chi phí bộ nhớ, tạo đối tượng, hủy đối tượng,... Tuy nhiên, trạng thái của các đối tượng được chia sẻ có thể cần được phân tích để xác định phần nào của trạng thái có thể được chia sẻ (và được lưu trữ vĩnh viễn bên trong đối tượng) và phần nào các phần của trạng thái cần được client của đối tượng chia sẻ cung cấp mỗi khi client truy cập đối tượng đó.

2.2. Mục đích của Flyweight Design Pattern

Mẫu thiết kế Flyweight giảm số lượng đối tượng cần thiết để xử lý bằng cách sắp xếp để nhiều đối tượng chia sẻ một số đối tượng khác mà chúng yêu cầu thay vì mỗi đối tượng có tài nguyên riêng.

Mục đích đằng sau mẫu thiết kế này là chia sẻ để hỗ trợ số lượng lớn các đối tượng chi tiết một cách hiệu quả. Để tạo điều kiện chia sẻ, một đối tượng dùng chung cần được thiết kế theo cách loại bỏ bất kỳ giá trị thuộc tính nào có thể ngăn không cho đối tượng đó được sử dụng bởi nhiều đối tượng khác. Các giá trị thuộc tính có thể được chia sẻ được gọi là dữ liệu nội tại và có thể nằm trong đối tượng được chia sẻ vì giá trị có liên quan đến tất cả các đối tượng được chia sẻ. Những giá trị không thể chia sẻ cấu thành dữ liệu bên ngoài và phải được mỗi đối tượng chia sẻ chuyển cho đối tượng được chia sẻ với mỗi lần sử dụng đối tượng.

2.3. Kiến trúc của Flyweight Design Pattern

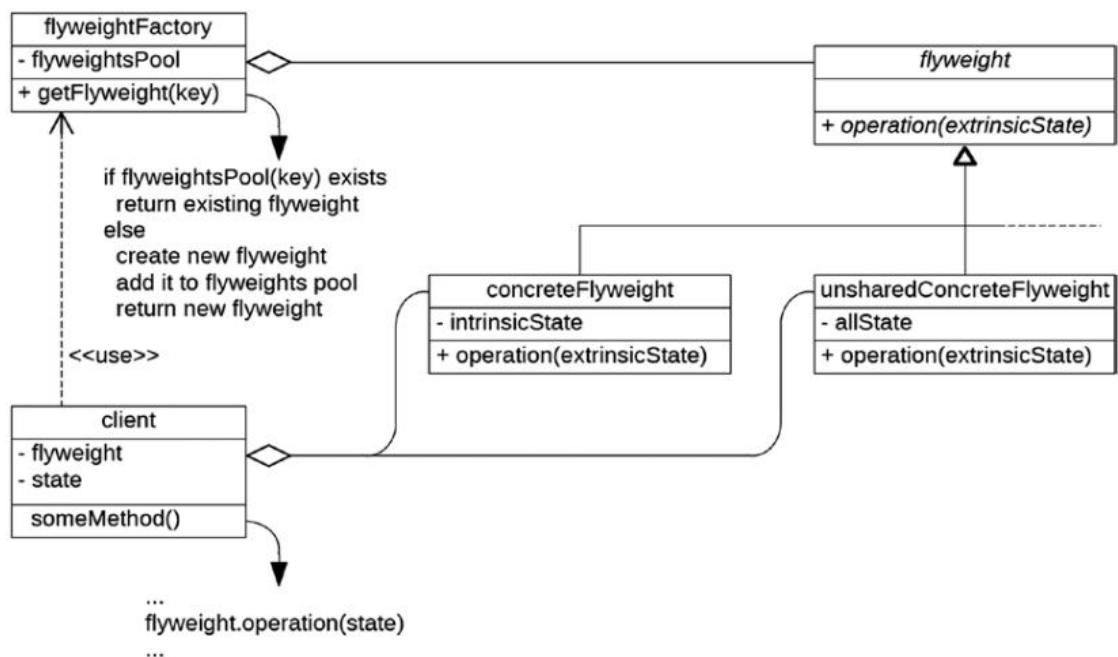
Flyweight sử dụng những thành phần sau đây:

- Flyweight: Khai báo một giao diện mà qua đó các flyweight có thể nhận và xử lý dữ liệu bên ngoài.
- ConcreteFlyweight: Triển khai giao diện Flyweight và thêm bộ nhớ cho dữ liệu nội tại nếu có. Đối tượng ConcreteFlyweight phải có thể chia sẻ được.

Bất kỳ dữ liệu nào nó lưu trữ phải là nội tại; nghĩa là nó phải độc lập với bối cảnh của đối tượng ConcreteFlyweight.

- UnsharedConcreteFlyweight: Không phải tất cả các lớp con Flyweight đều cần chia sẻ. Giao diện Flyweight cho phép chia sẻ; lớp này không thực thi nó. Các đối tượng UnsharedConcreteFlyweight thường có chứa các đối tượng ConcreteFlyweight ở tầng nào đó trong mô hình Flyweight.
- FlyweightFactory: Tạo và quản lý các đối tượng flyweight. Đảm bảo rằng các đối tượng flyweight được chia sẻ đúng cách. Khi một client yêu cầu một flyweight, đối tượng FlyweightFactory sẽ cung cấp một đối tượng hiện có hoặc tạo một đối tượng nếu không có đối tượng nào tồn tại.
- Client: Duy trì tham chiếu đến các flyweight. Tính toán hoặc lưu trữ dữ liệu bên ngoài của các flyweight.

Class diagram cho Flyweight pattern được mô tả trên:



Nhận xét:

FlyweightFactory giữ lại một nhóm các phiên bản của các đối tượng ConcreteFlyweight và UnsharedConcreteFlyweight, mỗi đối tượng sử dụng giao diện do Flyweight cung cấp. Việc triển khai phương thức getFlyweight sẽ xác định xem đã tồn tại phiên bản của flyweight theo giá trị khóa được yêu cầu chưa và nếu chưa tồn tại,

sẽ tạo một phiên bản flyweight mới sử dụng giá trị khóa đó và thêm nó vào nhóm flyweight, trả về cho người gọi đối tượng flyweight mới hoặc hiện có.

Những đối tượng ConcreteFlyweight giữ lại các thuộc tính đại diện cho trạng thái nội tại, đại diện cho thông tin có thể được chia sẻ giữa tất cả người dùng, nhưng nó chấp nhận thông tin trạng thái bên ngoài thông qua phương thức signature của phương thức công khai, yêu cầu người dùng cung cấp bất kỳ thông tin nào không thể được chia sẻ bởi tất cả người dùng của nó.

2.4. Ví dụ minh họa Flyweight Design Pattern

Đặt vấn đề

Trong một trò chơi giả lập chiến tranh, chúng ta lập trình cho trò chơi này có nhiều binh chủng: Lục Quân (Army), Hải Quân (Navy), Tuần Duyên (Coast Guard), Không Quân (Air Force), Thủy Quân Lục Chiến (Marine Corps). Mỗi khi người chơi có đủ ngân sách sẽ có thể tạo ra một binh lính mới. Thông tin của mỗi binh lính gồm có: id (mã định danh), star (cấp bậc trong quân ngũ), name (tên binh chủng). id có kiểu string, star có kiểu int, name có kiểu string. Như vậy name và id là hai thuộc tính tốn nhiều dung lượng nhất. Khi hiện thực trò chơi này và tạo ra rất nhiều binh lính cùng một lúc thì rất có thể sẽ xảy ra hiện tượng crash game trên các thiết bị có ít dung lượng RAM (Random Access Memory).

Giải quyết vấn đề

Chúng ta đều nhận thấy rằng khi các binh lính được tạo ra, họ sẽ có điểm chung là **Binh chủng (name)**. Như vậy thay vì cho mọi binh lính đều phải lưu riêng **Binh chủng (name)** riêng, ta sử dụng **Flyweight** để tận dụng lại thuộc tính **Binh chủng (name)** cho các binh lính. Thuộc tính **Binh chủng (name)** gọi là thuộc tính **nội tại (intrinsic state)**, **Mã định danh (id)** và **Cấp bậc (star)** gọi là thuộc tính **ngoại vi (extrinsic state)**.

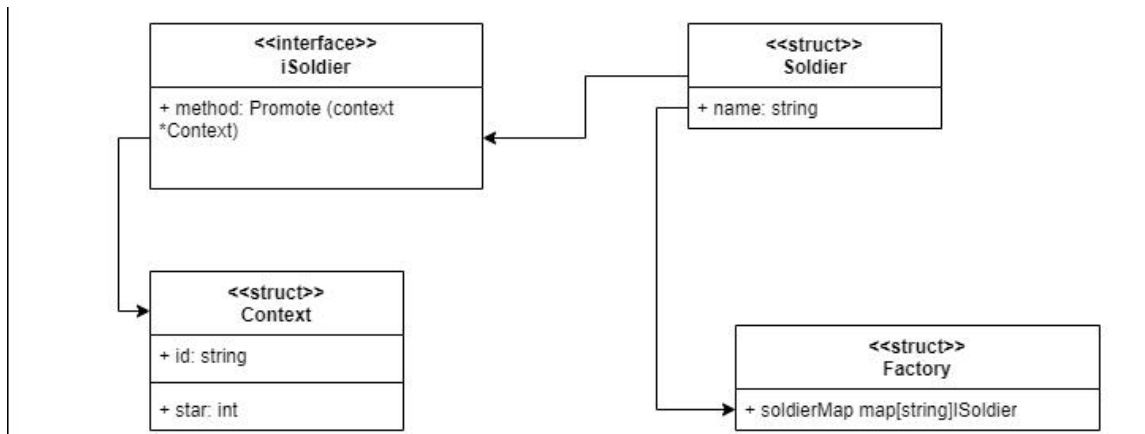


Image: Flyweight Pattern diagram example

Hiện thực bằng chương trình Golang:

- context.go

```

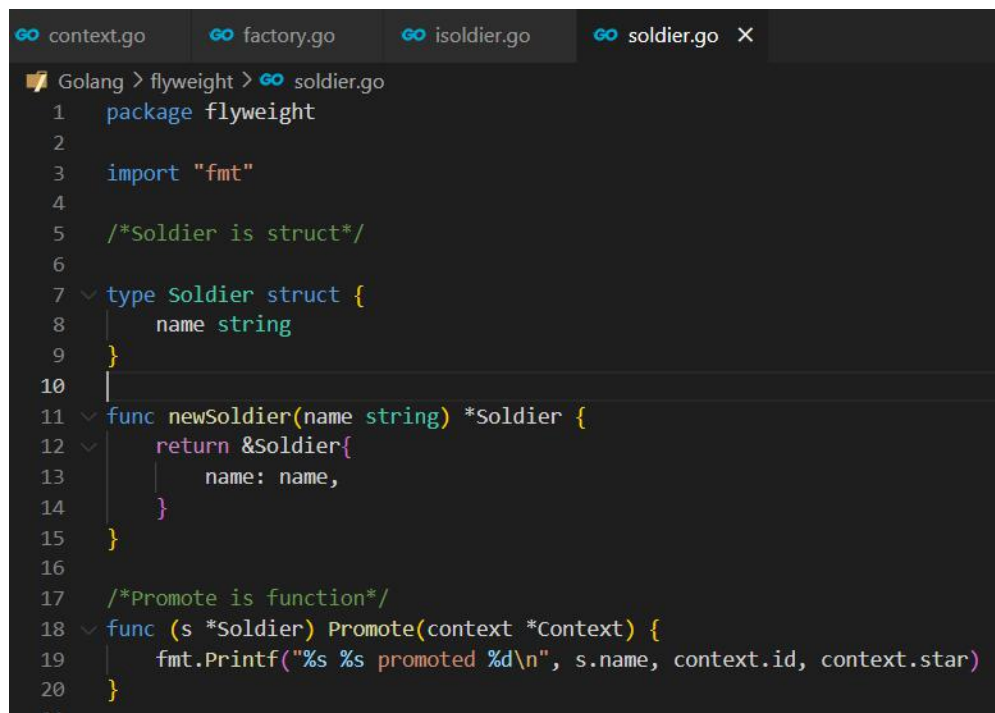
GO context.go X  GO factory.go  GO isoldier.go  GO soldier.go
Golang > flyweight > GO context.go
1  package flyweight
2
3  /*Context is struct*/
4  type Context struct {
5      id   string
6      star int
7  }
8
9  /*NewContext is function*/
10 func NewContext(id string, star int) *Context {
11     return &Context{
12         id:   id,
13         star: star,
14     }
15 }
  
```

- isoldier.go

```

GO context.go  GO factory.go  GO isoldier.go X  GO soldier.go
Golang > flyweight > GO isoldier.go
1  package flyweight
2
3  /*ISoldier is interface*/
4
5  type ISoldier interface {
6      Promote(context *Context)
7  }
8
  
```

- **soldier.go**

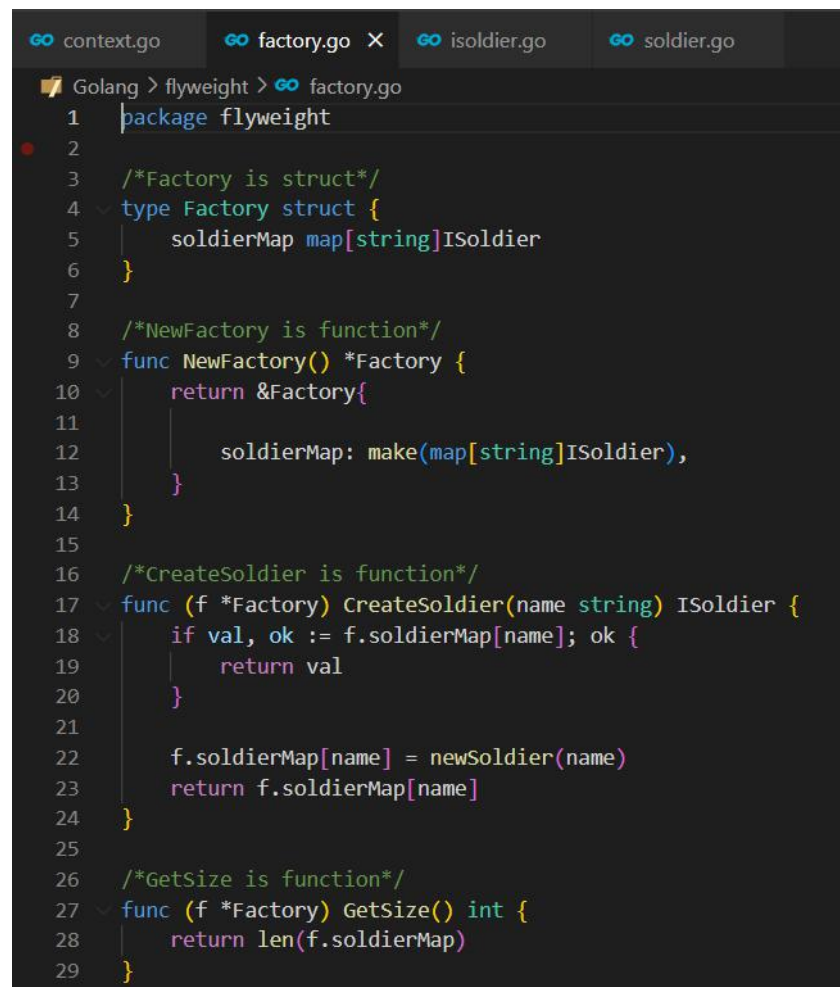


```

Golang > flyweight > soldier.go
1 package flyweight
2
3 import "fmt"
4
5 /*Soldier is struct*/
6
7 type Soldier struct {
8     name string
9 }
10
11 func newSoldier(name string) *Soldier {
12     return &Soldier{
13         name: name,
14     }
15 }
16
17 /*Promote is function*/
18 func (s *Soldier) Promote(context *Context) {
19     fmt.Printf("%s %s promoted %d\n", s.name, context.id, context.star)
20 }
21

```

- **factory.go**

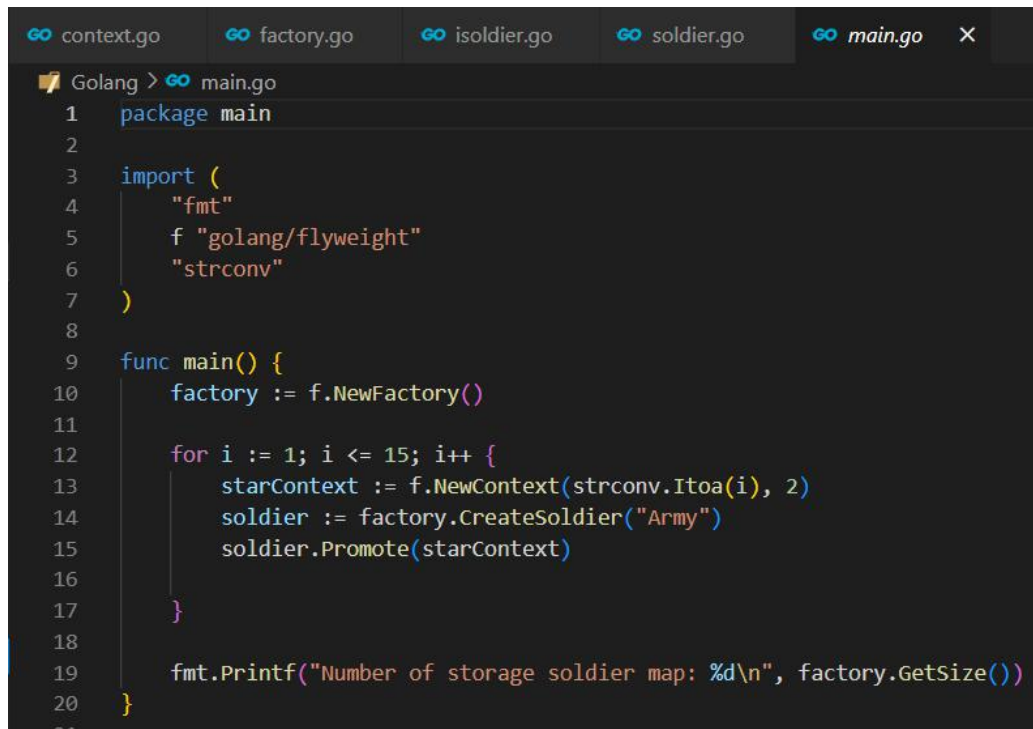


```

Golang > flyweight > factory.go
1 package flyweight
2
3 /*Factory is struct*/
4 type Factory struct {
5     soldierMap map[string]ISoldier
6 }
7
8 /*NewFactory is function*/
9 func NewFactory() *Factory {
10     return &Factory{
11         soldierMap: make(map[string]ISoldier),
12     }
13 }
14
15 /*CreateSoldier is function*/
16 func (f *Factory) CreateSoldier(name string) ISoldier {
17     if val, ok := f.soldierMap[name]; ok {
18         return val
19     }
20
21     f.soldierMap[name] = newSoldier(name)
22     return f.soldierMap[name]
23 }
24
25 /*GetSize is function*/
26 func (f *Factory) GetSize() int {
27     return len(f.soldierMap)
28 }
29

```

- **main.go**



```
1 package main
2
3 import (
4     "fmt"
5     f "golang/flyweight"
6     "strconv"
7 )
8
9 func main() {
10     factory := f.NewFactory()
11
12     for i := 1; i <= 15; i++ {
13         starContext := f.NewContext(strconv.Itoa(i), 2)
14         soldier := factory.CreateSoldier("Army")
15         soldier.Promote(starContext)
16     }
17
18     fmt.Printf("Number of storage soldier map: %d\n", factory.GetSize())
19 }
20
21
```

2.5. Ưu và nhược điểm của Flyweight Design Pattern

Ưu điểm:

Giảm yêu cầu lưu trữ của ứng dụng.

Giảm số lần tạo và hủy đối tượng xảy ra và do đó có thể cải thiện hiệu suất của ứng dụng.

Nhược điểm:

Tạo thêm chi phí runtime, vì các client có thể cần đặt trạng thái bên ngoài của các đối tượng.

Có thêm sự phức tạp trong ứng dụng, vì Flyweight Factory phải quản lý việc chia sẻ flyweights.

Gỡ lỗi và bảo trì có thể trở nên khó khăn hơn vì bất kỳ một đối tượng nào cũng có thể được chia sẻ giữa một số client.

2.6. Các trường hợp sử dụng Flyweight Design Pattern

Flyweight được sử dụng khi:

- Khi có một số lớn các đối tượng được ứng dụng tạo ra một cách lặp đi lặp lại.

- Khi việc tạo ra đối tượng đòi hỏi nhiều bộ nhớ và thời gian
- Khi muốn tái sử dụng đối tượng đã tồn tại thay vì phải tốn thời gian để tạo mới
- Khi nhóm đối tượng chứa nhiều đối tượng tương tự và hai đối tượng trong nhóm không khác nhau nhiều.

TỔNG KẾT

Trên đây là nội dung của bài báo cáo về Flyweight Design Pattern, một trong những Design Pattern phổ biến được sử dụng trong các ứng dụng cần tối ưu hóa việc sử dụng bộ nhớ. Chúng ta đã tìm hiểu về dữ liệu bên trong và bên ngoài, và rằng chúng ta có thể giảm số lượng đối tượng cần thiết trong một thiết kế nếu chúng ta có thể tạo các lớp chỉ giữ lại thông tin bên trong, để người dùng của lớp cung cấp bất kỳ thông tin bên ngoài nào có thể cần thiết. Mẫu này cung cấp cho chúng ta một cách khác để có thể chia sẻ các đối tượng giữa nhiều người dùng, sử dụng tài nguyên hiệu quả hơn, giảm mức tiêu thụ bộ nhớ và tối ưu hóa hiệu suất của các thành phần.

TÀI LIỆU THAM KHẢO

- [1] Dive into Design Pattern, Alexander Shvets, Refactoring.Guru
- [2] Design Patterns for Dummies, Steve Holzner, PhD,
<https://pdfroom.com/books/design-patterns-for-dummies-isbn-0471798541/KRd6oeGPgZp>
- [3] Gang of Four Software Design Patterns 4.0, Data & Object Factory, LLC,
<https://idoc.pub/documents/gang-of-four-design-patterns-40pdf-1d47x19qedl2>
- [4] Head First Design Patterns, Eric Freeman, Kathy Sierra, Bert Bates, Elisabeth Robson, <https://pdfroom.com/books/head-first-design-patterns/wW5mwjKkgYo>
- [5] Hunt, J. (2013). Scala Design Patterns: Patterns for Practical Reuse and Design, https://doi.org/10.1007/978-3-319-02192-8_24
- [6] McDonough, J. E. (2017). Object-Oriented Design with ABAP: A Practical Approach, https://doi.org/10.1007/978-1-4842-2838-8_24
- [7] Refactoring.Guru, <https://refactoring.guru/design-patterns>