

Samples - go

<https://g.co/gemini/share/70a16de99c6b>



Created with 2.5 Pro May 16, 2025 at 07:49 PM • Published July 13, 2025 at 03:18 PM

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git zapcore/buffered_write_syncer.go  
zapcore/buffered_write_syncer.go  
index 4b426a5..4e52826 100644  
--- zapcore/buffered_write_syncer.go  
+++ zapcore/buffered_write_syncer.go  
@@ -109,29 +109,6 @@ type BufferedWriteSyncer struct {  
     done      chan struct{} // closed when flushLoop has  
stopped  
}  
  
-func (s *BufferedWriteSyncer) initialize() {  
-    size := s.Size  
-    if size == 0 {  
-        size = _defaultBufferSize  
-    }  
-  
-    flushInterval := s.FlushInterval  
-    if flushInterval == 0 {
```

```

-         flushInterval = _defaultFlushInterval
-     }
-
-     if s.Clock == nil {
-         s.Clock = DefaultClock
-     }
-
-     s.ticker = s.Clock.NewTicker(flushInterval)
-     s.writer = bufio.NewWriterSize(s.WS, size)
-     s.stop = make(chan struct{})
-     s.done = make(chan struct{})
-     s.initialized = true
-     go s.flushLoop()
-}
-
// Write writes log data into buffer syncer directly, multiple
// Write calls will be batched,
// and log data will be flushed to disk when the buffer is full or
// periodically.
func (s *BufferedWriteSyncer) Write(bs []byte) (int, error) {
@@ -139,7 +116,26 @@ func (s *BufferedWriteSyncer)
Write(bs []byte) (int, error) {
    defer s.mu.Unlock()

    if !s.initialized {
-        s.initialize()
+        size := s.Size
+        if size == 0 {
+            size = _defaultBufferSize
+        }
+
+        flushInterval := s.FlushInterval
+        if flushInterval == 0 {
+            flushInterval = _defaultFlushInterval
+        }
+

```

```

+         if s.Clock == nil {
+             s.Clock = DefaultClock
+         }
+
+         s.ticker = s.Clock.NewTicker(flushInterval)
+         s.writer = bufio.NewWriterSize(s.WS, size)
+         s.stop = make(chan struct{})
+         s.done = make(chan struct{})
+         s.initialized = true
+         go s.flushLoop()
+     }

```

// To avoid partial writes from being flushed, we manually flush the existing buffer if:

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more

existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

****Instructions:****

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactorng with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- **Inline Method**

Justification:

- **Inline Method:** The method `initialize()` has been removed, and its entire body has been moved into the `Write()` method. The call to `s.initialize()` within `Write()` (specifically within the `if !s.initialized` block) has been replaced by the actual code of the `initialize()` method.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**
diff --git zapcore/buffered_write_syncer.go
zapcore/Rbuffered_write_syncer.go
similarity index 100%
rename from zapcore/buffered_write_syncer.go
rename to zapcore/Rbuffered_write_syncer.go
diff --git zapcore/buffered_write_syncer_test.go
zapcore/Rbuffered_write_syncer_test.go
similarity index 100%
rename from zapcore/buffered_write_syncer_test.go
rename to zapcore/Rbuffered_write_syncer_test.go
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

****Instructions:****

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



Based on the provided diff, the following refactoring type has been applied:

- **Rename Class**

Justification:

- **Rename Class:** The diff shows that the file `zapcore/buffered_write_syncer.go` has been renamed to `zapcore/Rbuffered_write_syncer.go`, and its corresponding test file `zapcore/buffered_write_syncer_test.go` has been renamed to `zapcore/Rbuffered_write_syncer_test.go`. In languages like Go, it's a common convention for a file to contain a primary type (struct, analogous to a class) that shares its name with the file (e.g., `buffered_write_syncer.go` likely defines a `BufferedWriteSyncer` struct). Renaming the file is a strong indication that the primary class/type it contains is being conceptually renamed. Although the diff indicates "similarity index 100%", meaning the internal type definition's name might not have changed *in this specific diff*, the act of renaming the file that encapsulates the class/type is considered part of the "Rename Class" refactoring. The synchronized renaming of the test file further supports that the core component (the "class") is what's being renamed.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- ****Diffs:****

diff --git zapcore/console_encoder.go

zapcore/console_encoder.go

index 98eea51..f1f3eb9 100644

--- zapcore/console_encoder.go

+++ zapcore/console_encoder.go

```
@@ -47,7 +47,7 @@ type consoleEncoder struct {  
    *jsonEncoder  
}
```

```
-// NewConsoleEncoder creates an encoder whose output is  
designed for human -
```

```
+// RNewConsoleEncoder creates an encoder whose output is  
designed for human -
```

```
// rather than machine - consumption. It serializes the core  
log entry data
```

```
// (message, level, timestamp, etc.) in a plain-text format and  
leaves the
```

```
// structured context as JSON.
```

```
@@ -55,7 +55,7 @@ type consoleEncoder struct {
```

```
// Note that although the console encoder doesn't use the  
keys specified in the
```

```
// encoder configuration, it will omit any element whose key is  
set to the empty
```

```
// string.
```

```
-func NewConsoleEncoder(cfg EncoderConfig) Encoder {
```

```
+func RNewConsoleEncoder(cfg EncoderConfig) Encoder {
```



```

        if cfg.ConsoleSeparator == "" {
            // Use a default delimiter of '\t' for backwards
compatibility
            cfg.ConsoleSeparator = "\t"
diff --git zapcore/console_encoder_bench_test.go
zapcore/console_encoder_bench_test.go
index a2a360f..bb86649 100644
--- zapcore/console_encoder_bench_test.go
+++ zapcore/console_encoder_bench_test.go
@@ -30,7 +30,7 @@ import (
func BenchmarkZapConsole(b *testing.B) {
    b.RunParallel(func(pb *testing.PB) {
        for pb.Next() {
-           enc :=
NewConsoleEncoder(humanEncoderConfig())
+           enc :=
RNewConsoleEncoder(humanEncoderConfig())
                enc.AddString("str", "foo")
                enc.AddInt64("int64-1", 1)
                enc.AddInt64("int64-2", 2)
diff --git zapcore/console_encoder_test.go
zapcore/console_encoder_test.go
index be8adc7..9b13d4f 100644
--- zapcore/console_encoder_test.go
+++ zapcore/console_encoder_test.go
@@ -68,7 +68,7 @@ func TestConsoleEncodeEntry(t
*ttesting.T) {

        cfg := testEncoderConfig()
        cfg.EncodeTime = RFC3339TimeEncoder
-       enc := NewConsoleEncoder(cfg)
+       enc := RNewConsoleEncoder(cfg)

        for _, tt := range tests {
            t.Run(tt.desc, func(t *testing.T) {

```

```

@@ -110,7 +110,7 @@ func TestConsoleSeparator(t *testing.T)
{
    }

    for _, tt := range tests {
-        console :=
NewConsoleEncoder(encoderTestEncoderConfig(tt.separator
))
+        console :=
RNewConsoleEncoder(encoderTestEncoderConfig(tt.separat
or))

        t.Run(tt.desc, func(t *testing.T) {
            entry := testEntry
            consoleOut, err := console.EncodeEntry(entry,
nil)
diff --git zapcore/encoder_test.go zapcore/encoder_test.go
index f89f489..7414ed9 100644
--- zapcore/encoder_test.go
+++ zapcore/encoder_test.go
@@ -531,7 +531,7 @@ func TestEncoderConfiguration(t
 *testing.T) {

    for i, tt := range tests {
        json := NewJSONEncoder(tt.cfg)
-        console := NewConsoleEncoder(tt.cfg)
+        console := RNewConsoleEncoder(tt.cfg)
        if tt.extra != nil {
            tt.extra(json)
            tt.extra(console)

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

****Instructions:****

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- **Rename Method**

Justification:

- **Rename Method:** The function `NewConsoleEncoder` in `zapcore/console_encoder.go` has been renamed to `RNewConsoleEncoder`. Its parameters (`cfg EncoderConfig`), return type (`Encoder`), and behavior (the function body, based on the unchanged parts shown and the nature of a rename) have been preserved. All call sites for this function, as seen in `console_encoder_bench_test.go`, `console_encoder_test.go`, and `encoder_test.go`, have been updated to use the new function name `RNewConsoleEncoder`.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **Diffs:**

diff --git config.go config.go

index e76e4e6..181aa40 100644

--- config.go

+++ config.go

@@ -130,7 +130,7 @@ func NewProductionEncoderConfig()

zapcore.EncoderConfig {

 FunctionKey: zapcore.OmitKey,

 MessageKey: "msg",

 StacktraceKey: "stacktrace",

- LineEnding: zapcore.DefaultLineEnding,

+ lineEnding: zapcore.DefaultLineEnding,

 EncodeLevel: zapcore.LowercaseLevelEncoder,

 EncodeTime: zapcore.EpochTimeEncoder,

 EncodeDuration: zapcore.SecondsDurationEncoder,

@@ -207,7 +207,7 @@ func NewDevelopmentEncoderConfig()

zapcore.EncoderConfig {

 FunctionKey: zapcore.OmitKey,

 MessageKey: "M",

 StacktraceKey: "S",

- LineEnding: zapcore.DefaultLineEnding,

+ lineEnding: zapcore.DefaultLineEnding,

 EncodeLevel: zapcore.CapitalLevelEncoder,

 EncodeTime: zapcore.ISO8601TimeEncoder,

 EncodeDuration: zapcore.StringDurationEncoder,

diff --git zapcore/console_encoder.go

zapcore/console_encoder.go

index f1f3eb9..d3899ab 100644

--- zapcore/console_encoder.go

+++ zapcore/console_encoder.go

@@ -125,7 +125,7 @@ func (c consoleEncoder)

EncodeEntry(ent Entry, fields []Field) (*buffer.Buffer,

 line.AppendString(ent.Stack)

```

    }

    -   line.AppendString(c.LineEnding)
    +   line.AppendString(c.lineEnding)
    return line, nil
}

diff --git zapcore/encoder.go zapcore/encoder.go
index 0446254..cbe9d7d 100644
--- zapcore/encoder.go
+++ zapcore/encoder.go
@@ -339,7 +339,7 @@ type EncoderConfig struct {
     FunctionKey   string `json:"functionKey"
yaml:"functionKey"`
     StacktraceKey string `json:"stacktraceKey"
yaml:"stacktraceKey"`
     SkipLineEnding bool  `json:"skipLineEnding"
yaml:"skipLineEnding"`
    -   LineEnding   string `json:"lineEnding" yaml:"lineEnding"`
    +   lineEnding   string `json:"lineEnding" yaml:"lineEnding"`
    // Configure the primitive representations of common
complex types. For
    // example, some users may want all time.Times serialized
as floating-point
    // seconds since epoch, while others may prefer ISO8601
strings.
@@ -358,6 +358,14 @@ type EncoderConfig struct {
    ConsoleSeparator string `json:"consoleSeparator"
yaml:"consoleSeparator"`
}

+func (e *EncoderConfig) LineEnding() string {
+    return e.lineEnding
+}
+
+func (e *EncoderConfig) SetLineEnding(lineEnding string) {

```

```

+     e.lineEnding = lineEnding
+}
+
// ObjectEncoder is a strongly-typed, encoding-agnostic
interface for adding a
// map- or struct-like object to the logging context. Like
maps, ObjectEncoders
// aren't safe for concurrent use (though typical use shouldn't
require locks).
diff --git zapcore/json_encoder.go zapcore/json_encoder.go
index 9685169..c9c5447 100644
--- zapcore/json_encoder.go
+++ zapcore/json_encoder.go
@@ -80,9 +80,9 @@ func NewJSONEncoder(cfg
EncoderConfig) Encoder {

func newJSONEncoder(cfg EncoderConfig, spaced bool)
*jsonEncoder {
    if cfg.SkipLineEnding {
-        cfg.LineEnding = ""
-    } else if cfg.LineEnding == "" {
-        cfg.LineEnding = DefaultLineEnding
+        cfg.lineEnding = ""
+    } else if cfg.lineEnding == "" {
+        cfg.lineEnding = DefaultLineEnding
    }

    // If no EncoderConfig.NewReflectedEncoder is provided
by the user, then use default
@@ -423,7 +423,7 @@ func (enc *jsonEncoder)
EncodeEntry(ent Entry, fields []Field) (*buffer.Buffer,
    final.AddString(final.StacktraceKey, ent.Stack)
    }
    final.buf.AppendByte(',')
-    final.buf.AppendString(final.LineEnding)
+    final.buf.AppendString(final.lineEnding)

```

```
ret := final.buf  
putJSONEncoder(final)
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the

transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

****Instructions:****

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



The following refactoring types have been applied:

- **Rename Field**
- **Encapsulate Field**

Justification:

1. **Rename Field:**

- The field `LineEnding` in the `zapcore.EncoderConfig` struct (defined in `zapcore/encoder.go`) has been renamed to `lineEnding`.

Diff



```
--- zapcore/encoder.go
+++ zapcore/encoder.go
@@ -339,7 +339,7 @@ type EncoderConfig struct {
     FunctionKey      string `json:"functionKey" yaml:"functionKey"`
     StacktraceKey     string `json:"stacktraceKey" yaml:"stacktraceKey"`
     SkipLineEnding    bool   `json:"skipLineEnding" yaml:"skipLineEnding"`
-    LineEnding        string `json:"lineEnding" yaml:"lineEnding"`
+    lineEnding        string `json:"lineEnding" yaml:"lineEnding"`
```

- Usages of this field in struct initializations (in `config.go`) and direct accesses within the `zapcore` package (in `console_encoder.go`, `json_encoder.go`) have been updated from `LineEnding` to `lineEnding`. For example, in `config.go`:

Diff



```
-    LineEnding:    zapcore.DefaultLineEnding,
+    lineEnding:    zapcore.DefaultLineEnding,
```

And in `zapcore/json_encoder.go`:

Diff



```
-    final.buf.AppendString(final.LineEnding)
+    final.buf.AppendString(final.lineEnding)
```

2. Encapsulate Field:

- The field `LineEnding` in `zapcore.EncoderConfig` was previously exported (public, due to the uppercase first letter). After being renamed to `lineEnding` (lowercase first letter), it became unexported (package-private in Go).
- Public getter (`LineEnding()`) and setter (`SetLineEnding(lineEnding string)`) methods have been introduced in `zapcore/encoder.go` to control access to this now-unexported `lineEnding` field from outside the `zapcore` package.

```
--- zapcore/encoder.go
+++ zapcore/encoder.go
@@ -358,6 +358,14 @@ type EncoderConfig struct {
     ConsoleSeparator string `json:"consoleSeparator" yaml:"consoleSeparator"`
 }

+func (e *EncoderConfig) LineEnding() string {
+    return e.lineEnding
+}
+
+func (e *EncoderConfig) SetLineEnding(lineEnding string) {
+    e.lineEnding = lineEnding
+}
```

This change restricts direct access to the field from other packages and provides controlled access through methods, which is the definition of encapsulating a field.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**
diff --git zapcore/error.go zapcore/error.go
index c40df13..7e5546e 100644
--- zapcore/error.go
+++ zapcore/error.go
@@ -107,7 +107,7 @@ func (errs errArray)
 MarshalLogArray(arr ArrayEncoder) error {
     return nil
 }

-var _errArrayElemPool = pool.New(func() *errArrayElem {
+var _RerrArrayElemPool = pool.New(func() *errArrayElem {
     return &errArrayElem{}
 })
```

```

@@ -117,7 +117,7 @@ var _errArrayElemPool = pool.New(func()
*errArrayElem {
    type errArrayElem struct{ err error }

    func newErrArrayElem(err error) *errArrayElem {
-       e := _errArrayElemPool.Get()
+       e := _RerrArrayElemPool.Get()
        e.err = err
        return e
    }
@@ -132,5 +132,5 @@ func (e *errArrayElem)
MarshalLogObject(enc ObjectEncoder) error {

    func (e *errArrayElem) Free() {
        e.err = nil
-       _errArrayElemPool.Put(e)
+       _RerrArrayElemPool.Put(e)
    }

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

****Instructions:****

1. Begin your response with a bullet-point list of the refactoring type(s) applied.

2. Then, briefly justify each identified refactorng with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- **Rename Field**

Justification:

- **Rename Field:** The package-level variable `_errArrayElemPool` in `zapcore/error.go` has been renamed to `_RerrArrayElemPool`. This variable, which holds a pool instance, acts as a private static field for the package. All references to this variable within the file (in the `newErrArrayElem` function and the `Free` method of `errArrayElem`) have been updated to use the new name `_RerrArrayElemPool`.

Diff



```
--- zapcore/error.go
+++ zapcore/error.go
@@ -107,7 +107,7 @@ func (errs errArray) MarshalLogArray(arr ArrayEncoder)
     return nil
 }

-var _errArrayElemPool = pool.New(func() *errArrayElem {
+var _RerrArrayElemPool = pool.New(func() *errArrayElem {
     return &errArrayElem{}
 })

@@ -117,7 +117,7 @@ var _errArrayElemPool = pool.New(func() *errArrayElem
type errArrayElem struct{ err error }

func newErrArrayElem(err error) *errArrayElem {
-    e := _errArrayElemPool.Get()
+    e := _RerrArrayElemPool.Get()
    e.err = err
    return e
}
```

```
@@ -132,5 +132,5 @@ func (e *errArrayElem) MarshalLogObject(enc ObjectEncoder) error {

func (e *errArrayElem) Free() {
    e.err = nil
-    _errArrayElemPool.Put(e)
+    _RerrArrayElemPool.Put(e)
}
```

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**
diff --git zapcore/lhooked.go zapcore/lhooked.go
new file mode 100644
index 0000000..ada49f0
--- /dev/null
+++ zapcore/lhooked.go
@@ -0,0 +1,5 @@
+package zapcore
+
+type lhooked interface {
+    Level() Level
+}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

****Instructions:****

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- **Extract Interface**

Justification:

- **Extract Interface:** The diff shows the creation of a new file `zapcore/Ihooked.go` which defines a new interface named `Ihooked`. This interface specifies a single method, `Level() Level`. The refactoring "Extract Interface" describes the process of creating a new interface from existing (or intended to be common) method signatures in classes. While the diff only shows the newly created interface and not the class(es) from which the `Level()` method might have been extracted, the creation of this new interface `Ihooked` is the direct outcome of such a refactoring.

Diff



```
--- /dev/null
+++ zapcore/Ihooked.go
@@ -0,0 +1,5 @@
+package zapcore
+
+type Ihooked interface {
+    Level() Level
+}
```

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **Diffs:**

diff --git zapcore/level.go zapcore/level.go

index f3e166d..ec85de4 100644

--- zapcore/level.go

+++ zapcore/level.go

@@ -68,7 +68,7 @@ const (

// levels.

func ParseLevel(text string) (Level, error) {

var level Level

- err := level.UnmarshalText([]byte(text))

+ err := level.RUnmarshalText([]byte(text))

return level, err

}

@@ -161,13 +161,13 @@ func (l Level) MarshalText() ([]byte, error) {

return []byte(l.String()), nil

}

-// UnmarshalText unmarshals text to a level. Like MarshalText,
UnmarshalText

+// RUnmarshalText unmarshals text to a level. Like
MarshalText, RUnmarshalText

// expects the text representation of a Level to drop the -
Level suffix (see

// example).

//

```

// In particular, this makes it easy to configure logging levels
using YAML,
// TOML, or JSON files.
-func (l *Level) UnmarshalText(text []byte) error {
+func (l *Level) RUnmarshalText(text []byte) error {
    if l == nil {
        return errUnmarshalNilLevel
    }
@@ -201,7 +201,7 @@ func (l *Level) unmarshalText(text
[]byte) bool {

// Set sets the level for the flag.Value interface.
func (l *Level) Set(s string) error {
-    return l.UnmarshalText([]byte(s))
+    return l.RUnmarshalText([]byte(s))
}

// Get gets the level for the flag.Getter interface.
diff --git zapcore/level_test.go zapcore/level_test.go
index d0beadc..36527fc 100644
--- zapcore/level_test.go
+++ zapcore/level_test.go
@@ -71,7 +71,7 @@ func TestLevelText(t *testing.T) {
    }

    var unmarshaled Level
-    err := unmarshaled.UnmarshalText([]byte(tt.text))
+    err := unmarshaled.RUnmarshalText([]byte(tt.text))
    assert.NoError(t, err, `Unexpected error
unmarshaling text %q to level.`, tt.text)
    assert.Equal(t, tt.level, unmarshaled, `Text %q
unmarshaled to an unexpected level.`, tt.text)
}
@@ -81,7 +81,7 @@ func TestLevelText(t *testing.T) {
    t.Run("unmarshal warning compatibility", func(t
*testing.T) {

```

```

var unmarshaled Level
input := []byte("warning")
-   err := unmarshaled.UnmarshalText(input)
+   err := unmarshaled.RUnmarshalText(input)
    assert.NoError(t, err, `Unexpected error
unmarshaling text %q to level.`, string(input))
    assert.Equal(t, WarnLevel, unmarshaled, `Text %q
unmarshaled to an unexpected level.`, string(input))
})
@@ -124,7 +124,7 @@ func TestCapitalLevelsParse(t
*testing.T) {
    }
    for _, tt := range tests {
        var unmarshaled Level
-       err := unmarshaled.UnmarshalText([]byte(tt.text))
+       err := unmarshaled.RUnmarshalText([]byte(tt.text))
        assert.NoError(t, err, `Unexpected error
unmarshaling text %q to level.`, tt.text)
        assert.Equal(t, tt.level, unmarshaled, `Text %q
unmarshaled to an unexpected level.`, tt.text)
    }
@@ -155,7 +155,7 @@ func TestWeirdLevelsParse(t *testing.T)
{
    }
    for _, tt := range tests {
        var unmarshaled Level
-       err := unmarshaled.UnmarshalText([]byte(tt.text))
+       err := unmarshaled.RUnmarshalText([]byte(tt.text))
        assert.NoError(t, err, `Unexpected error
unmarshaling text %q to level.`, tt.text)
        assert.Equal(t, tt.level, unmarshaled, `Text %q
unmarshaled to an unexpected level.`, tt.text)
    }
@@ -173,13 +173,13 @@ func TestLevelNils(t *testing.T) {
    _, _ = l.MarshalText() // should panic
}, "Expected to panic when marshalling a nil level.")

```

```

-   err := l.UnmarshalText([]byte("debug"))
+   err := l.RUnmarshalText([]byte("debug"))
    assert.Equal(t, errUnmarshalNilLevel, err, "Expected to
error unmarshalling into a nil Level.")
}

func TestLevelUnmarshalUnknownText(t *testing.T) {
    var l Level
-   err := l.UnmarshalText([]byte("foo"))
+   err := l.RUnmarshalText([]byte("foo"))
    assert.ErrorContains(t, err, "unrecognized level",
"Expected unmarshaling arbitrary text to fail.")
}

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

****Instructions:****

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- **Rename Method**

Justification:

- **Rename Method:** The method `UnmarshalText` defined on the `*Level` type in `zapcore/level.go` has been renamed to `RUnmarshalText`.

Diff



```
--- zapcore/level.go
+++ zapcore/level.go
@@ -161,13 +161,13 @@ func (l Level) MarshalText() ([]byte, error) {
     return []byte(l.String()), nil
 }

-// UnmarshalText unmarshals text to a level. Like MarshalText, UnmarshalText
+// RUnmarshalText unmarshals text to a level. Like MarshalText, RUnmarshalText
// expects the text representation of a Level to drop the -Level suffix (e.g., "info"
// example).
//
// In particular, this makes it easy to configure logging levels using YAML, TOML, or JSON files.
-func (l *Level) UnmarshalText(text []byte) error {
+func (l *Level) RUnmarshalText(text []byte) error {
```

The method's signature (receiver `(l *Level)`, parameter `(text []byte)`, and return type `error`) and its behavior (the method body) have been preserved. All call sites of this method have been updated to the new name `RUnmarshalText` in both `zapcore/level.go` (e.g., within the `ParseLevel` and `Set` functions) and in the test file `zapcore/level_test.go`. For example, in `zapcore/level.go`:

Diff



```
-     err := level.UnmarshalText([]byte(text))
+     err := level.RUnmarshalText([]byte(text))
```

And in `zapcore/level_test.go` :

Diff



```
-      err := unmarshaled.UnmarshalText([]byte(tt.text))
+      err := unmarshaled.RUnmarshalText([]byte(tt.text))
```

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**
diff --git basic/main.go basic/main.go
index 6d86997..5bc818c 100644
--- basic/main.go
+++ basic/main.go
@@ -1,25 +1,24 @@
package main

import (
-   "net/http"
-
+   "github.com/gin-gonic/gin"
+   "net/http"
)

var db = make(map[string]string)

-func setupRouter() *gin.Engine {
+func main() {
    // Disable Console Color
    // gin.DisableConsoleColor()
```



```

-   r := gin.Default()
+   r2 := gin.Default()

    // Ping test
-   r.GET("/ping", func(c *gin.Context) {
+   r2.GET("/ping", func(c *gin.Context) {
        c.String(http.StatusOK, "pong")
    })

    // Get user value
-   r.GET("/user/:name", func(c *gin.Context) {
+   r2.GET("/user/:name", func(c *gin.Context) {
        user := c.Params.ByName("name")
        value, ok := db[user]
        if ok {
@@ -36,7 +35,7 @@ func setupRouter() *gin.Engine {
    //     "foo": "bar",
    //     "manu": "123",
    //}))
-   authorized := r.Group("/", gin.BasicAuth(gin.Accounts{
+   authorized := r2.Group("/", gin.BasicAuth(gin.Accounts{
        "foo": "bar", // user:foo password:bar
        "manu": "123", // user:manu password:123
    }))
@@ -63,12 +62,7 @@ func setupRouter() *gin.Engine {
        c.JSON(http.StatusOK, gin.H{"status": "ok"})
    }
    })
-
-   return r
-}
-
-func main() {
-   r := setupRouter()
+   r := r2
    // Listen and Server in 0.0.0.0:8080

```

```

    r.Run(":8080")
}
diff --git basic/main_test.go basic/main_test.go
index 5eb8524..af7aebf 100644
--- basic/main_test.go
+++ basic/main_test.go
@@ -9,7 +9,61 @@ import (
)

func TestPingRoute(t *testing.T) {
-   router := setupRouter()
+   // Disable Console Color
+   // gin.DisableConsoleColor()
+   r := gin.Default()
+
+   // Ping test
+   r.GET("/ping", func(c *gin.Context) {
+       c.String(http.StatusOK, "pong")
+   })
+
+   // Get user value
+   r.GET("/user/:name", func(c *gin.Context) {
+       user := c.Params.ByName("name")
+       value, ok := db[user]
+       if ok {
+           c.JSON(http.StatusOK, gin.H{"user": user,
+"value": value})
+       } else {
+           c.JSON(http.StatusOK, gin.H{"user": user,
+"status": "no value"})
+       }
+   })
+
+   // Authorized group (uses gin.BasicAuth() middleware)
+   // Same than:
+   // authorized := r.Group("/")

```

```

+ // authorized.Use(gin.BasicAuth(gin.Credentials{
+ //     "foo": "bar",
+ //     "manu": "123",
+ //}))
+ authorized := r.Group("/", gin.BasicAuth(gin.Accounts{
+     "foo": "bar", // user:foo password:bar
+     "manu": "123", // user:manu password:123
+ }))
+
+ /* example curl for /admin with basicauth header
+     Zm9vOmJhcG== is base64("foo:bar")
+
+     curl -X POST \
+     http://localhost:8080/admin \
+     -H 'authorization: Basic Zm9vOmJhcG==' \
+     -H 'content-type: application/json' \
+     -d '{"value":"bar"}'
+ */
+ authorized.POST("admin", func(c *gin.Context) {
+     user := c.MustGet(gin.AuthUserKey).(string)
+
+     // Parse JSON
+     var json struct {
+         Value string `json:"value" binding:"required"`
+     }
+
+     if c.Bind(&json) == nil {
+         db[user] = json.Value
+         c.JSON(http.StatusOK, gin.H{"status": "ok"})
+     }
+ })
+
+ router := r

w := httptest.NewRecorder()
req, _ := http.NewRequest("GET", "/ping", nil)

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

****Instructions:****

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- **Inline Method**

Justification:

- **Inline Method:** The `setupRouter()` function has been eliminated, and its body has been directly incorporated into its former call sites.
 - In `basic/main.go`, the call `r := setupRouter()` within the `main` function has been replaced by the entire logic of the `setupRouter` function. The `main` function now directly contains the router setup code previously encapsulated in `setupRouter`.

Diff



```

--- basic/main.go
+++ basic/main.go
@@ -5,16 +5,15 @@ import (
    )

    var db = make(map[string]string)

- func setupRouter() *gin.Engine {
+ func main() {
    // Disable Console Color
    // gin.DisableConsoleColor()
-    r := gin.Default()
+    r2 := gin.Default() // formerly r in setupRouter

    // Ping test
-    r.GET("/ping", func(c *gin.Context) { // ... and other r. calls
+    r2.GET("/ping", func(c *gin.Context) { // ... and other r2. call
// ... (body of setupRouter) ...
-    return r
- }
-
- func main() {
-    r := setupRouter()
+    r := r2 // assignment of the inlined router
    // Listen and Server in 0.0.0.0:8080
    r.Run(":8080")
}

```

- Similarly, in `basic/main_test.go`, the call `router := setupRouter()` within the `TestPingRoute` function has been replaced by the full body of the `setupRouter` function.

Diff



```

--- basic/main_test.go
+++ basic/main_test.go
@@ -9,7 +9,61 @@ import (
    )

    func TestPingRoute(t *testing.T) {
-        router := setupRouter()
+        // Disable Console Color
+        // gin.DisableConsoleColor()
+        r := gin.Default()

```

```

+
+ // Ping test
+ r.GET("/ping", func(c *gin.Context) {
+     c.String(http.StatusOK, "pong")
+ })
+ // ... (rest of the setupRouter body copied here) ...
+ router := r

```

This replacement of method calls with the method's body is the definition of the "Inline Method" refactoring.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```

- **Diffs:**
diff --git grpc/example1/gen/helloworld/v1/IRouter.go
grpc/example1/gen/helloworld/v1/IRouter.go
new file mode 100644
index 0000000..8156207
--- /dev/null
+++ grpc/example1/gen/helloworld/v1/IRouter.go
@@ -0,0 +1,7 @@
+package v1
+
+import "github.com/gin-gonic/gin"
+
+type RouterGroup interface {
+    GET(relativePath string, handlers ...gin.HandlerFunc)
+    gin.IRoutes
+}

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new

version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

****Instructions:****

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- **Extract Interface**

Justification:

- **Extract Interface:** The diff shows the creation of a new file `grpc/example1/gen/helloworld/v1/IRouter.go`, which defines a new interface named `RouterGroup`. This interface specifies a single method signature: `GET(relativePath string, handlers ...gin.HandlerFunc) gin.IRoutes`. The refactoring "Extract Interface" describes the creation of a new interface based on existing method(s) in a class (or in this Go context, often from existing struct types or other interfaces). While the diff only shows the newly created interface and not the specific struct(s) or source from which the `GET` method signature was derived, the creation of this new `RouterGroup` interface to define a contract for routing capabilities is the direct outcome of such a refactoring. The `GET` method signature is characteristic of routing groups in web frameworks like Gin, suggesting this interface is intended to abstract that behavior.

```
--- /dev/null
+++ grpc/example1/gen/helloworld/v1/IRouter.go
@@ -0,0 +1,7 @@
+package v1
+
+import "github.com/gin-gonic/gin"
+
+type RouterGroup interface {
+    GET(relativePath string, handlers ...gin.HandlerFunc) gin.IRoutes
+}
```

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**
diff --git forward-proxy/main.go forward-proxy/main.go
index 1260547..3eaa827 100644
--- forward-proxy/main.go
+++ forward-proxy/main.go
@@ -33,7 +33,11 @@ func ForwardMid(c *gin.Context) {
        return
    }
    defer resp.Body.Close()
-    copyHeader(c.Writer.Header(), resp.Header)
+    for k, vv := range resp.Header {
+        for _, v2 := range vv {
+            c.Writer.Header().Add(k, v2)
+        }
+    }
    c.Writer.WriteHeader(resp.StatusCode)
```

```

        io.Copy(c.Writer, resp.Body)
        c.Abort()
@@ -44,14 +48,6 @@ func ForwardMid(c *gin.Context) {
    c.Next()
}

-func copyHeader(dst, src http.Header) {
-    for k, vv := range src {
-        for _, v := range vv {
-            dst.Add(k, v)
-        }
-    }
-}
-
func Reverse(c *gin.Context) {
    remote, _ := url.Parse("http://xxx.xxx.xxx")
    proxy := httputil.NewSingleHostReverseProxy(remote)

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

****Instructions:****

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactorng with reference to the specific code changes.
3. Only include refactorings from the list above.

4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- **Inline Method**

Justification:

- **Inline Method:** The function `copyHeader(dst, src http.Header)` has been removed, and its body has been directly substituted into its call site within the `ForwardMid` function.
 - The original call in `ForwardMid` :

Go



```
copyHeader(c.Writer.Header(), resp.Header)
```

- Was replaced with the logic of `copyHeader` :

Diff



```
-      copyHeader(c.Writer.Header(), resp.Header)
+      for k, vv := range resp.Header {
+          for _, v2 := range vv {
+              c.Writer.Header().Add(k, v2)
+          }
+      }
```