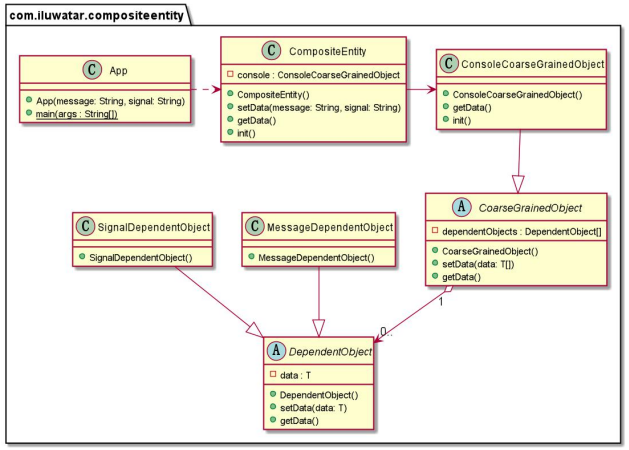
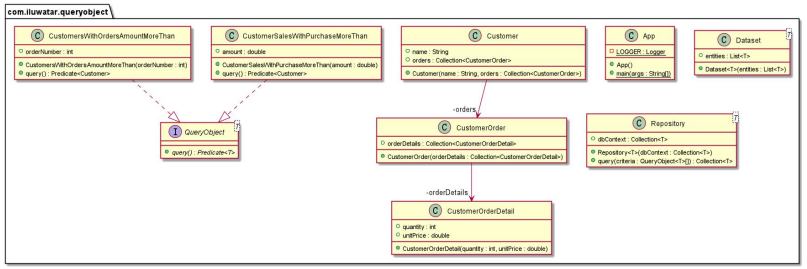
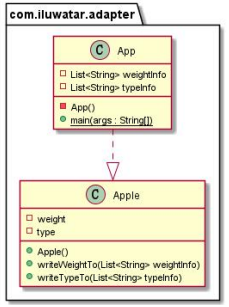
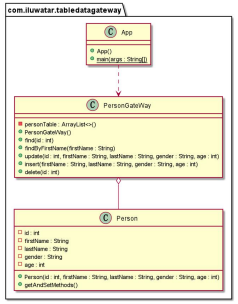
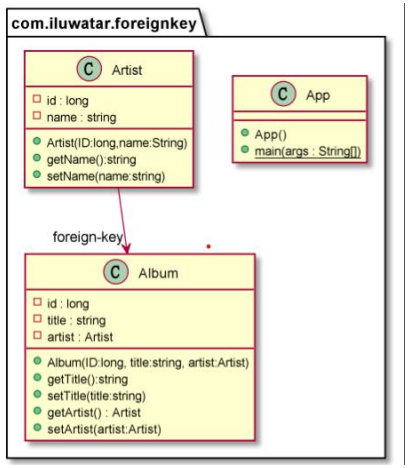




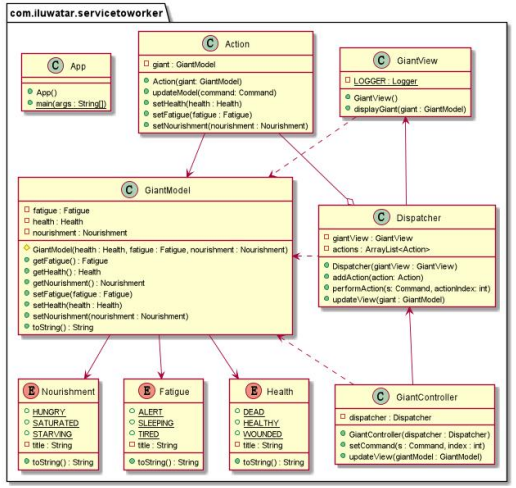


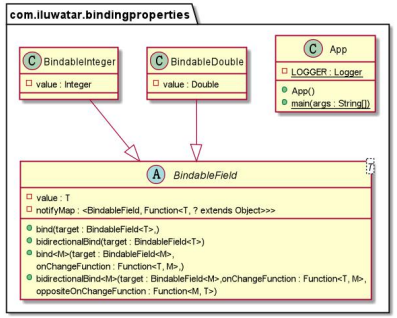




NoLSP

Selected project: java-design-patterns, JsonPath





https://github.com/iluwatar/java-design-patterns
https://github.com/json-path/JsonPath

庄湛 11811721
张湲 11811702
陈宇 11811203
刘洺琛 11810614
罗叶安 11810616
魏源泰 11811719

Project Introduction

java-design-patterns	
Watch	3.9k
Star	67k
Fork	21k

Design patterns are the best formalized practices a programmer can use to solve common problems when designing an application or system.

Design patterns can speed up the development process by providing tested, proven development paradigms.

Reusing design patterns help prevent subtle issues that cause major problems, and it also improves code readability for coders and architects who are familiar with the patterns.

Issue id	Design Pattern	PR id
1269	Data Transfer Hash Pattern	1704
1268	Composite Entity Pattern	1705
1308	Query Object Pattern	1714
1261	Collecting Parameter Pattern	1715
1262	Service to Worker Pattern	1718
1318	Table Data Gateway Pattern	1720
1296	Foreign Key Mapping Pattern	1723
1298	Identity Field Pattern	1723
1293	Dependent Mapping Pattern	1761
67	Binding Properties Pattern	1776

Project Introduction

JsonPath	
Watch	280
Star	5.9k
Fork	1.1k

A Java DSL for reading JSON documents.

Issue id	Design Pattern	PR id
628	JsonPath read using parsed document loses configuration	696
707	when use only one escape char in string, user may get a wrong string.	709

Path Examples

Operators

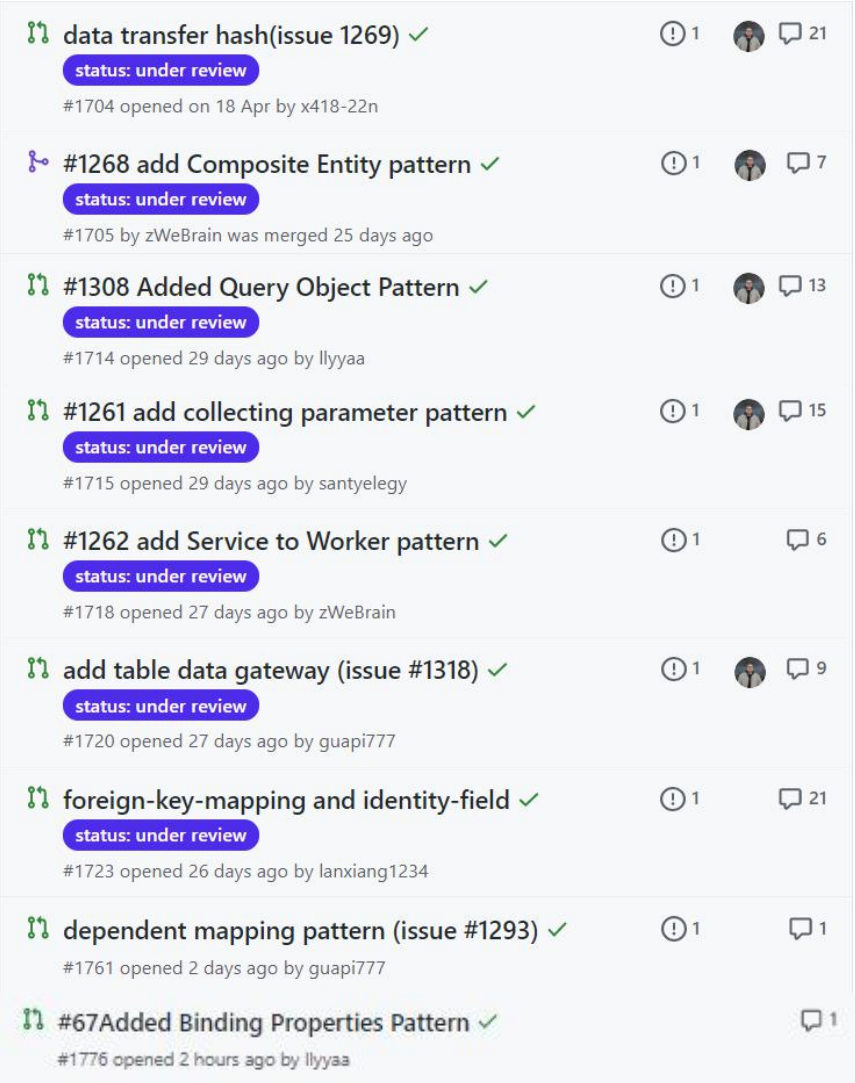
Operator	Description
`$`	The root element to query. This starts all path expressions.
`@`	The current node being processed by a filter predicate.
`*`	Wildcard. Available anywhere a name or numeric are required.
`..`	Deep scan. Available anywhere a name is required.
`.<name>`	Dot-notated child
`['<name>' (, '<name>')]`	Bracket-notated child or children
`[<number> (, <number>)]`	Array index or indexes
`[start:end]`	Array slice operator
`[?(<expression>)]`	Filter expression. Expression must evaluate to a boolean value.

```
{
    "store": {
        "book": [
            {
                "category": "reference",
                "author": "Nigel Rees",
                "title": "Sayings of the Century",
                "price": 8.95
            },
            {
                "category": "fiction",
                "author": "Evelyn Waugh",
                "title": "Sword of Honour",
                "price": 12.99
            },
            {
                "category": "fiction",
                "author": "Herman Melville",
                "title": "Moby Dick",
                "isbn": "0-553-21311-3",
                "price": 8.99
            },
            {
                "category": "fiction",
                "author": "J. R. R. Tolkien",
                "title": "The Lord of the Rings",
                "isbn": "0-395-19395-8",
                "price": 22.99
            }
        ],
        "bicycle": {
            "color": "red",
            "price": 19.95
        }
    },
    "expensive": 10
}
```

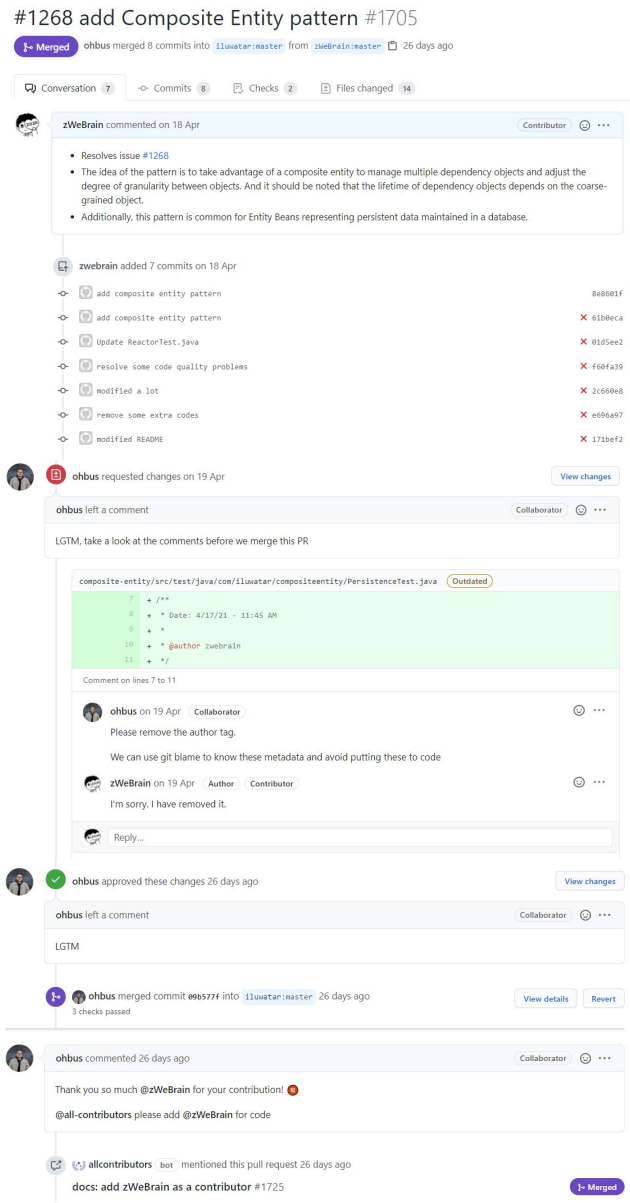
JsonPath (click link to try)	Result
$.store.book[*].author	The authors of all books
$..author	All authors
$.store.*	All things, both books and bicycles
$.store..price	The price of everything
$..book[2]	The third book
$..book[-2]	The second to last book
$..book[0,1]	The first two books
$..book[:2]	All books from index 0 (inclusive) until index 2 (exclusive)
$..book[1:2]	All books from index 1 (inclusive) until index 2 (exclusive)
$..book[-2:]	Last two books
$..book[2:]	Book number two from tail
$..book[?(@.isbn)]	All books with an ISBN number
$.store.book[?(@.price < 10)]	All books in store cheaper than 10
$..book[?(@.price <= $['expensive'])]	All books in store that are not "expensive"
$..book[?(@.author =~ /.*REES/i)]	All books matching regex (ignore case)
$..*	Give me every thing
$..book.length()	The number of books

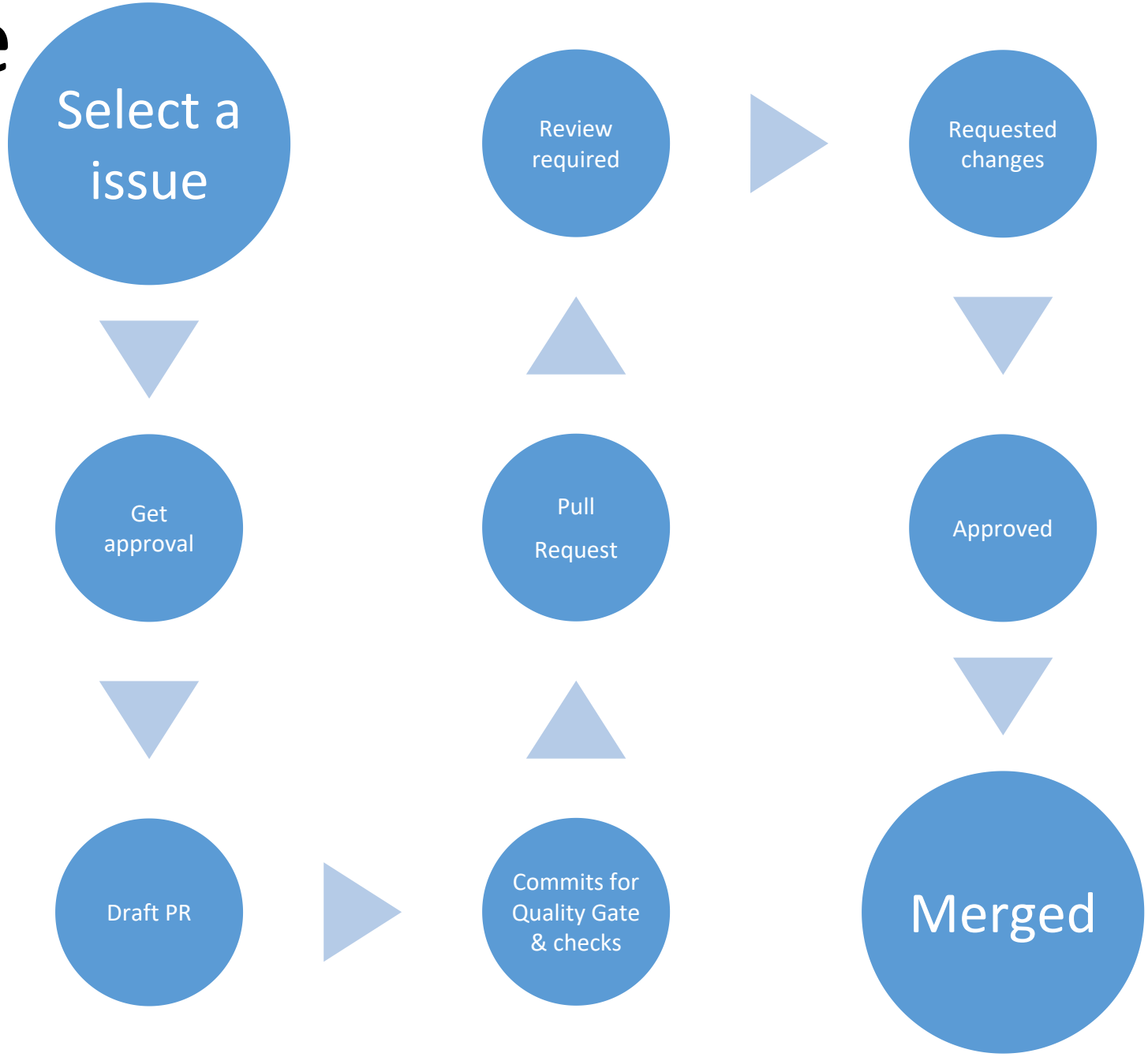
Timeline

Time	Issues Implemented Subpart 1	Issues Implemented Subpart 2
Week 2	Read the contribution wiki and code	Read the contribution wiki and code
Week 3	Work on JsonPath issue #628	Work on Data Transfer Hash Pattern #1269
Week 4	Work on Composite Entity Pattern #1269	Work on Data Transfer Hash Pattern #1269
Week 5	Work on Query Object Pattern #1308	Work on Collecting Parameter Pattern #1261
Week 6	Deal with problems raised by the author	Deal with problems raised by the author
Week 7	Work on Service to Worker Pattern #1262	Work on Table Data Gateway Pattern #1318
Week 8	Work on Service to Worker Pattern #1262	Work on Table Data Gateway Pattern #1318
Week 9	Work on Foreign Key Mapping #1296	Work on Identity Field Pattern #1298
Week 10	Deal with problems raised by the author	Deal with problems raised by the author
Week 11	Deal with problems raised by the author	Deal with problems raised by the author
Week 12	Work on Foreign Key Mapping #1296	Work on Binding Properties Pattern #67
Week 13	Deal with problems raised by the author	Deal with problems raised by the author
Week 14	Work on Dependent Mapping #1293	Work on JsonPath issue #707
Week 15	Prepare final release of the project	Prepare final release of the project



Timeline: Example

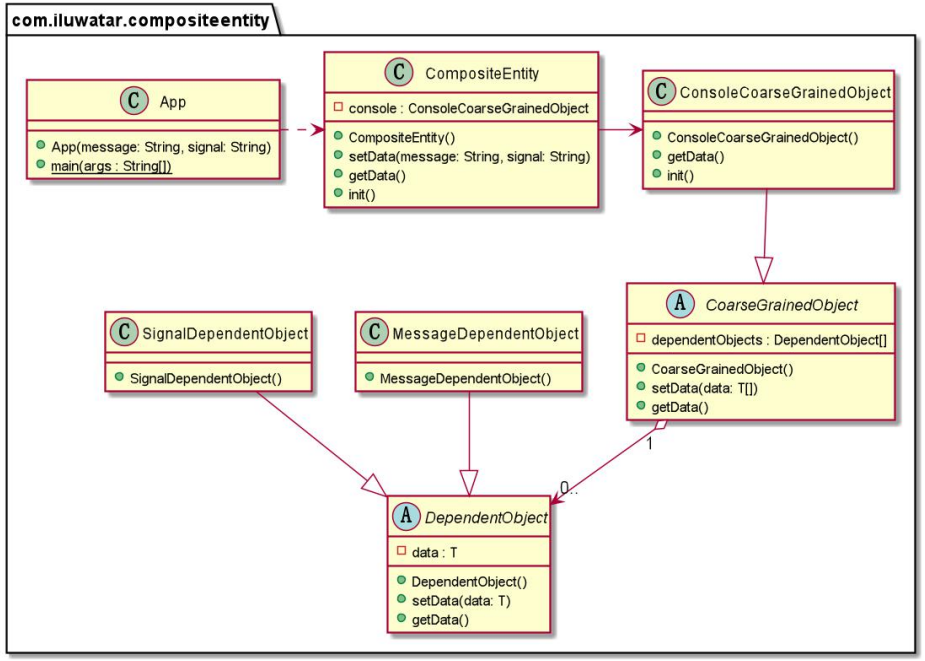




Important Issues: Composite Entity Pattern

1268	Composite Entity Pattern	1705

..		
etc	task: Add Composite Entity pattern (#1705)	26 days ago
src	task: Add Composite Entity pattern (#1705)	26 days ago
README.md	fix: Fixed pages showing up in wrong language (#1752)	3 days ago
pom.xml	task: Add Composite Entity pattern (#1705)	26 days ago



README.md

layout	title	folder	permalink	categories	language	tags
pattern	Composite Entity	composite-entity	/patterns/composite-entity/	Structural	en	Enterprise Integration Pattern

Intent

It is used to model, represent, and manage a set of persistent objects that are interrelated, rather than representing them as individual fine-grained entities.

Explanation

Real world example

> For a console, there may be many interfaces that need to be managed and controlled. Using the composite entity pattern, dependent objects such as messages and signals can be combined together and controlled using a single object.

In plain words

> Composite entity pattern allows a set of related objects to be represented and managed by a unified object.

Programmatic Example

We need a generic solution for the problem. To achieve this, let's introduce a generic Composite Entity Pattern.

....

Applicability

Use the Composite Entity Pattern in the following situation:

- You want to manage multiple dependency objects through one object to adjust the degree of granularity between objects. At the same time, the lifetime of dependency objects depends on a coarse-grained object.

Credits

- Composite Entity Pattern in wikipedia

https://github.com/iluwatar/java-design-patterns/tree/master/composite-entity

Important Issues: bugs about escape char

707	when use only one escape char in string, user may get a wrong string.	709

```java
/**
 * CS304 (manually written) Issue link:
 * https://github.com/json-path/JsonPath/issues/707
 */
public class issue707 extends BaseTest{
@Test
    public void Test_Escape_character1() {
        String JSON="{\"\\data\": \"a \\ b\"}";
        assertThat((String)(((JsonPath.parse(JSON)
                .read("$.\\data"))))).isEqualTo("a \\ b");

    }

    @Test
    public void Test_Escape_character2() {
        String JSON="{\"a\": [{\"data\"  : \"a \\ b\"}]}";
        assertThat((String)(((net.minidev.json.JSONArray)(JsonPath.parse(JSON)
                .read("$.a[*].data"))).get(0))).isEqualTo("a \\ b");
    }
}
```

x418-22n commented 9 days ago

fix bug : when use only one escape char in string, user may get a wrong string.
for example:
in this case

```java
String JSON="{\"a\": [{\"data\"  : \"a \\ b\"}]}";
System.out.println((String)(((net.minidev.json.JSONArray)(JsonPath.parse(JSON)
        .read("$.a[*].data"))).get(0)))
```

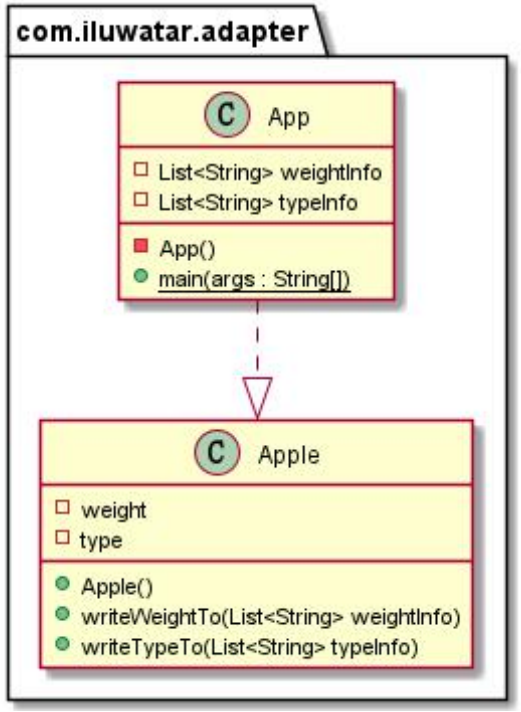
Before, we wouldn't get `a \ b` but `a b`. I think it is wrong.
New it will return `a \ b`,which is correct.

https://github.com/cs304-spring2021/finalpre-nolsp/blob/master/JsonPath/issue%23707

Important Issues: Collecting Parameter Pattern

1261	Collecting Parameter Pattern	1715

etc	Java design pattern code
src	Java design pattern code
README.md	Java design pattern code
pom.xml	Java design pattern code



README.md

layout	title	folder	permalink	categories
pattern	Collecting parameter	collecting-parameter	/patterns/collecting-parameter/	Structural

Intent

A Collecting Parameter is an object that you pass to methods in order to collect information from those methods. This pattern is often coupled with Composed Method

In the CollectingParameter idiom a collection (list, map, etc.) is passed repeatedly as a parameter to a method which adds items to the collection.

Explanation

Real world example

> Consider that You have a single bulky method that accumulates information to a local variable. A Collecting Parameter may also be passed to methods on multiple objects Collecting parameter pattern lets you make the accumulating method into some simple method.

....

Applicability

Use the collecting parameter pattern when

- You have a single bulky method that accumulates information to a local variable

Consequences:

Collecting parameter pattern can:

- Helps transform bulky methods into smaller, simpler methods.
- make resulting code run faster.

Real world examples

- junit.framework.TestResult

Credits

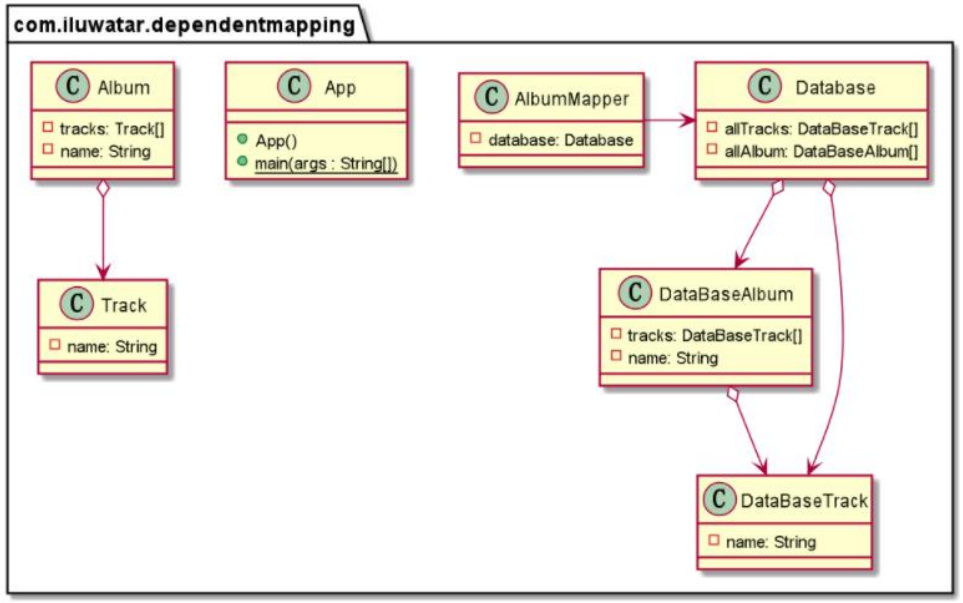
- Design Patterns: Elements of Reusable Object-Oriented Software
- J2EE Design Patterns
- Head First Design Patterns: A Brain-Friendly Guide
- Refactoring to Patterns
- Collecting Parameter

https://github.com/cs304-spring2021/finalpre-nolsp/tree/master/java-design-pattern/collecting-parameter

Important Issues: Dependent Mapping Pattern

1293	Dependent Mapping Pattern	1761

..	
etc	Java design pattern code
src	Java design pattern code
README.md	Java design pattern code
pom.xml	Java design pattern code



README.md

layout	title	folder	permalink	categories	tags
pattern	Dependent Mapping	dependent-mapping	/patterns/dependent-mapping/	Behavioral	Database

Intent

The basic idea behind Dependent Mapping is that one class (the dependent) relies upon some other class (the owner) for its database persistence. Each dependent can have only one owner and must have one owner.

A dependent may itself be the owner of another dependent. In this case the owner of the first dependent is also responsible for the persistence of the second dependent. You can have a whole hierarchy of dependents controlled by a single primary owner.

Writing and saving of dependents is left to the owner, and there are no outside references, updates to the dependents can be handled through deletion and insertion. Thus, if you want to update the collection of dependents you can safely delete all rows that link to the owner and then reinsert all the dependents.

Class diagram

Applicability

You use Dependent Mapping when you have an object that's only referred to by one other object, which usually occurs when one object has a collection of dependents. Dependent Mapping is a good way of dealing with the awkward situation where the owner has a collection of references to its dependents but there's no back pointer. Providing that the many objects don't need their own identity, using Dependent Mapping makes it easier to manage their persistence.

For Dependent Mapping to work there are a number of preconditions.

- A dependent must have exactly one owner.
- There must be no references from any object other than the owner to the dependent.

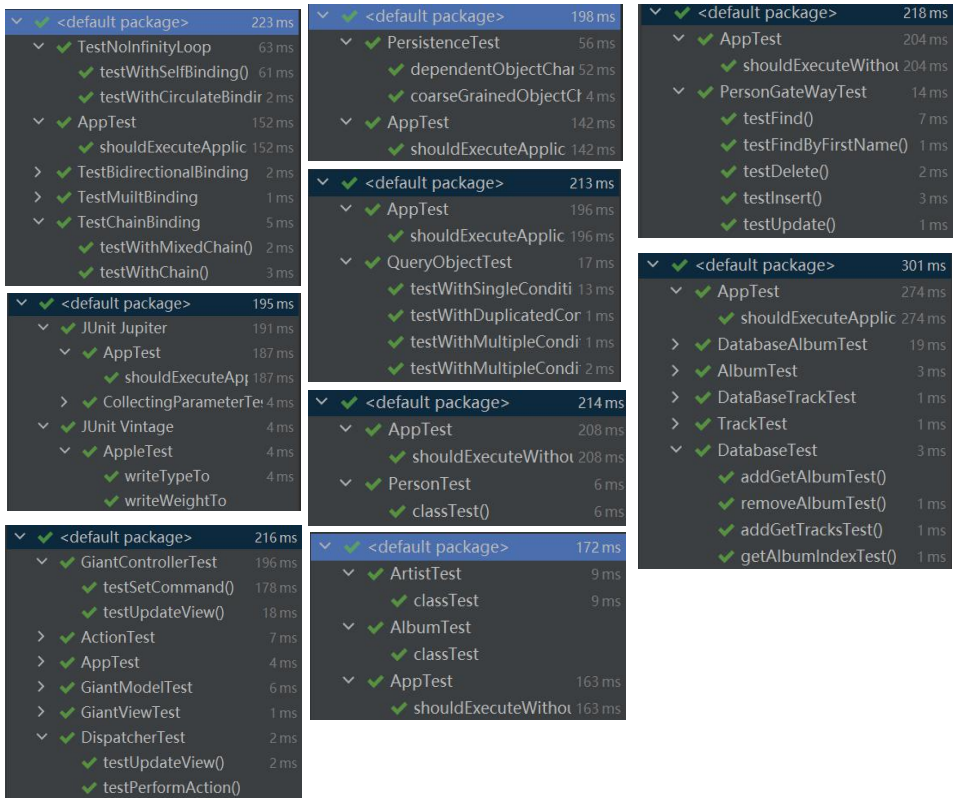
Credits

- Dependent Mapping Pattern-1
- Dependent Mapping Pattern-2

https://github.com/cs304-spring2021/finalpre-nolsp/tree/master/java-design-pattern/dependent-mapping

Testing

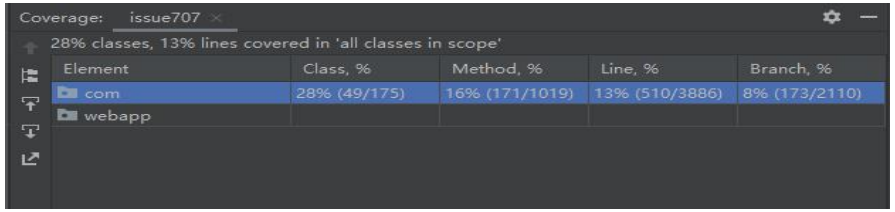
Screenshot



Coverage

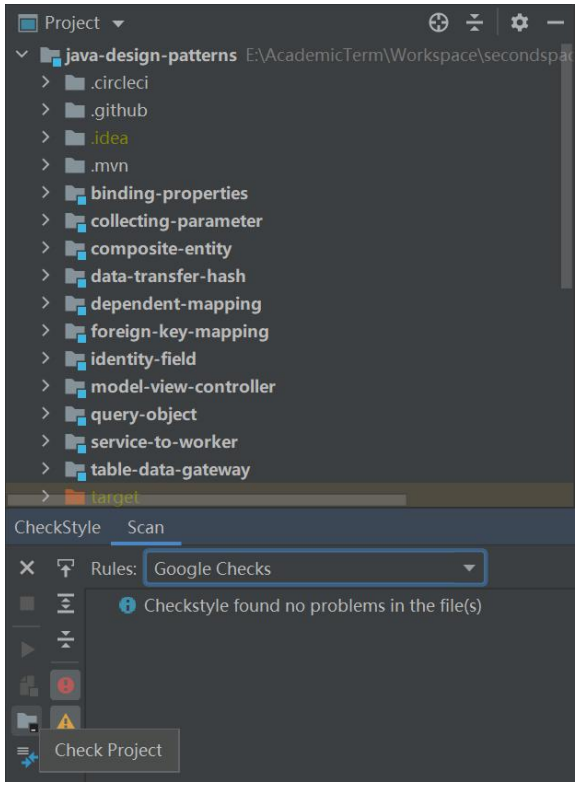
Design Pattern	Test	Class, %	Method, %	Line, %
Data Transfer Hash Pattern	4	100% (4/4)	100% (8/8)	100% (26/26)
Composite Entity Pattern	3	100% (7/7)	100% (12/12)	96% (32/33)
Query Object Pattern	5	100% (8/8)	100% (15/15)	100% (67/67)
Collecting Parameter Pattern	4	100% (2/2)	100% (7/7)	100% (34/34)
Service to Worker Pattern	13	100% (7/7)	100% (27/27)	100% (74/74)
Table Data Gateway Pattern	6	100% (3/3)	95% (19/20)	92% (65/70)
Foreign Key Mapping Pattern	3	100% (3/3)	100% (10/10)	100% (29/29)
Identity Field Pattern	1	100% (2/2)	100% (6/6)	100% (19/19)
Dependent Mapping Pattern	15	100% (7/7)	100% (23/23)	99% (99/100)
Binding Properties Pattern	7	100% (5/5)	100% (28/28)	100% (95/95)

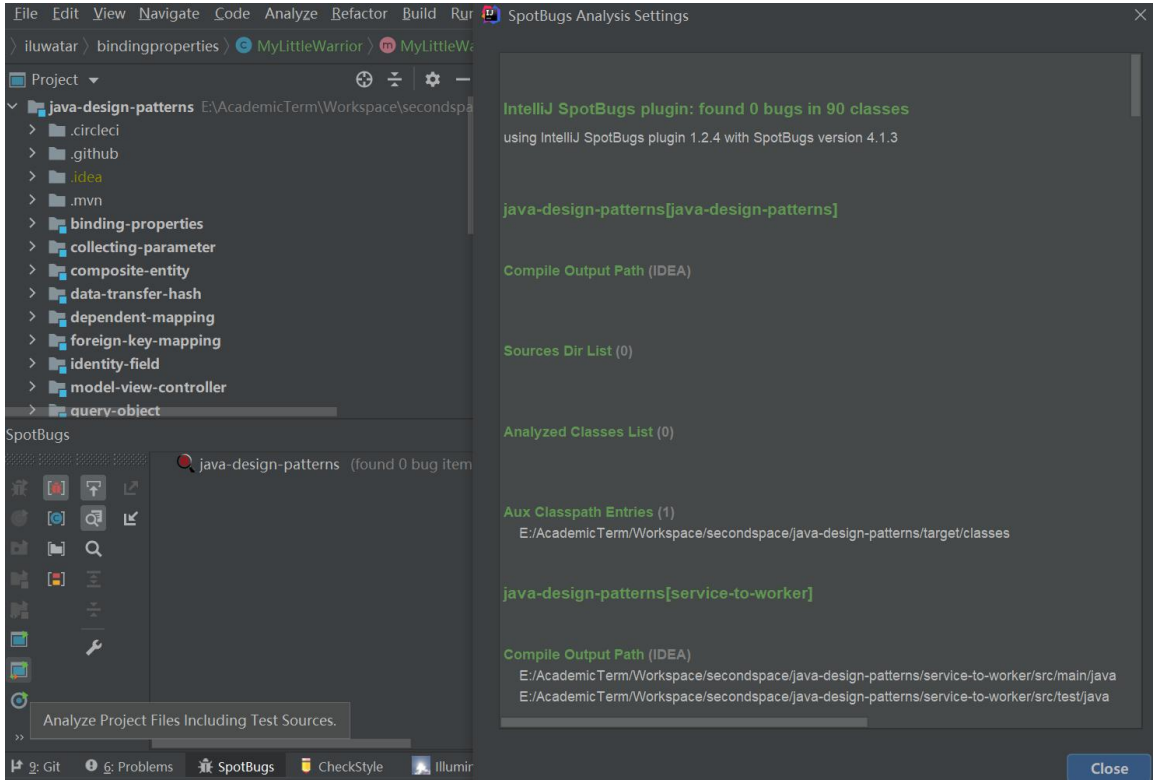
For JsonPath



Test technique: All are manual test

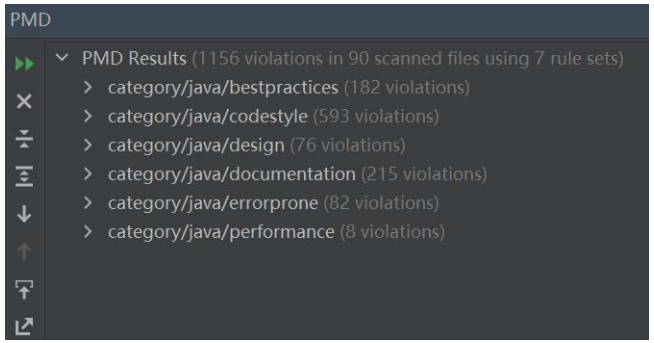
Static Analysising





1. Checkstyle (google):
   no problems
2. Findbugs:
   0 bugs in 90 class
3. PMD:
   no warnings
4. Quality Gate (bot)：
   0 Bugs
   0 Vulnerability
   0 Security Hotspots
   few Code Smells

All checks have passed!



Extension	Count	Size SUM	Size MIN	Size MAX	Size AVG	Lines	Lines MIN	Lines MAX	Lines AVG	Lines CODE
gitignore (*GITIGNORE files*)	1x	0kB	0kB	0kB	0kB	3	3	3	3	2
java (*Java classes*)	90x	115kB	0kB	4kB	1kB	4002	9	122	44	1986
md (*MD files*)	11x	30kB	0kB	7kB	2kB	813	21	179	73	546
puml (*PUML files*)	11x	10kB	0kB	2kB	0kB	421	12	86	38	398
txt (*Text files*)	1x	0kB	0kB	0kB	0kB	6	6	6	6	6
ucls (*UCLS files*)	1x	3kB	3kB	3kB	3kB	53	53	53	53	53
xml (*XML configuration file*)	21x	101kB	0kB	42kB	4kB	1856	6	465	88	1767

Results demonstrating

Design Pattern	Intent	Scenarios
Composite Entity	It is used to model, represent, and manage a set of persistent objects that are interrelated, rather than representing them as individual fine-grained entities.	For a console, there may be many interfaces that need to be managed and controlled. Using the composite entity pattern, dependent objects such as messages and signals can be combined together and controlled using a single object.
Collecting Parameter	A Collecting Parameter is an object that you pass to methods in order to collect information from those methods.	Consider that you have a single bulky method that accumulates information to a local variable. A Collecting Parameter may also be passed to methods on multiple objects.
Dependent Mapping	The basic idea behind Dependent Mapping is that one class (the dependent) relies upon some other class (the owner) for its database persistence. Each dependent can have only one owner and must have one owner.	You use Dependent Mapping when you have an object that’ s only referred to by one other object, which usually occurs when one object has a collection of dependents. Dependent Mapping is a good way of dealing with the awkward situation where the owner has a collection of references to its dependents but there’s no back pointer.

Conclusions

1. For JsonPath, we learned a lot about the ideas and concepts of Json design, and deeply understood the principle of Json files and the convenience that this project can bring to Json files. Finally, we fix two bugs and provided some tests.

2. For java-design-patterns, we learned a lot about new design patterns and their applications. These design patterns will have many applications in industry. Finally, we provide a total of ten design patterns, each with detailed documentation, class diagrams, templates, and rich test scenarios.

Future Work

1. By contributing to open source projects, we have learned a lot of collaborative development skills and become proficient in using Git. At the same time, working in groups makes our projects go smoothly, and it turns out that working in groups works better.

2. Our difficulty is to understand the source code of JsonPath and the new design pattern of java-design-patterns. Especially the latter, for each design pattern, we need to look up a lot of data, consider a lot of scenarios.

3. We hope we can make some contribution to these open source projects, and try to achieve different design patterns in different ways with more vivid examples.

4.In the future, we will maintain the bug fixed on JsonPath and keep the design pattern updated on java-design-patterns. We actually have a lot of interesting problems, but we solve them through our own efforts.