

Turning Chained Branching into the Chain of Rule Objects



Zoran Horvat
CEO AT CODING HELMET

@zoranh75 <http://codinghelmet.com>



Pluralsight > src > com > codinghelmet > moreoojava > Demo

LifetimeWarranty.java < VoidWarranty.java < Article.java < Demo.java < StatusEqualityRule.java < Action.java < ClaimingRule.java < TimeLimitedWarranty.java

Project 1: Project

1: Structure

2: Favorites

14

15 private void claimWarranty(Article article, DeviceStatus status, Optional<LocalDate> sensorFailureDate) {

16 LocalDate today = LocalDate.now();

17

18 StatusEqualityRule.match(

19 DeviceStatus.allFine(),

20 () -> this.claimMoneyBack(article, today))

21 .orElse(StatusEqualityRule.match(

22 DeviceStatus.notOperational(),

23 () -> {

24 this.claimMoneyBack(article, today);

25 this.claimExpress(article, today);

26 }))

27 .applicableTo(status)

28 .ifPresent(action -> action.apply());

29

30 Runnable allFineAction = () ->

31 this.claimMoneyBack(article, today);

32

33 Runnable notOperationalAction = () -> {

34 this.claimMoneyBack(article, today);

35 this.claimExpress(article, today);

36 };

37

38 if (status.equals(DeviceStatus.allFine())) {

39 allFineAction.run();

40 } else if (status.equals(DeviceStatus.notOperational())) {

41 notOperationalAction.run();

Demo > claimWarranty()

Find Debug TODO Terminal Messages Run Event Log

Composite Rule

Pluralsight > src > com > codinghelmet > moreoojava > Demo

LifetimeWarranty.java < VoidWarranty.java < Article.java < Demo.java < StatusEqualityRule.java < Action.java < ClaimingRule.java < TimeLimitedWarranty.java

Project 1: Project

1: Structure

2: Favorites

14

15 private void claimWarranty(Article article, DeviceStatus status, Optional<LocalDate> sensorFailureDate) {

16 LocalDate today = LocalDate.now();

17

18 StatusEqualityRule.match(

19 DeviceStatus.allFine(),

20 () -> this.claimMoneyBack(article, today))

21 .orElse(StatusEqualityRule.match(

22 DeviceStatus.notOperational(),

23 () -> {

24 this.claimMoneyBack(article, today);

25 this.claimExpress(article, today);

26 }))

27 .applicableTo(status)

28 .ifPresent(action -> action.apply());

29

30 Runnable allFineAction = () ->

31 this.claimMoneyBack(article, today);

32

33 Runnable notOperationalAction = () -> {

34 this.claimMoneyBack(article, today);

35 this.claimExpress(article, today);

36 };

37

38 if (status.equals(DeviceStatus.allFine())) {

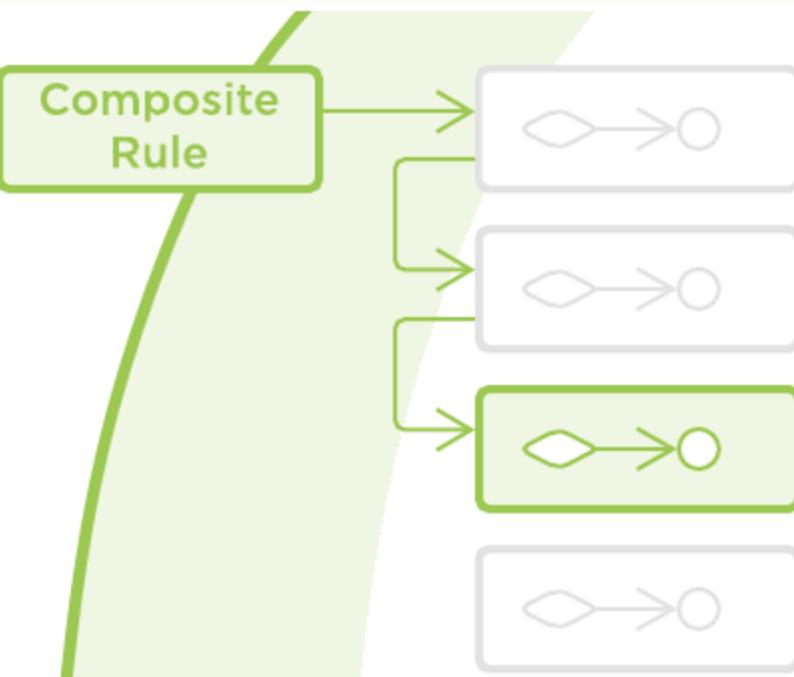
39 allFineAction.run();

40 } else if (status.equals(DeviceStatus.notOperational())) {

41 notOperationalAction.run();

Demo > claimWarranty()

Find Debug TODO Terminal Messages Run Event Log



Pluralsight > src > com > codinghelmet > moreoojava > ChainedRule

Main

LifetimeWarranty.java VoidWarranty.java Article.java Demo.java ChainedRule.java StatusEqualityRule.java Action.java ClaimingRule.java

Project

Ant Build

m Maven

```
1: Project
2: Favorites
3: Structure
4: I: Structure
5: Favorites
6: Structure
7: I: Structure
8: Favorites
9: Structure
10: I: Structure
11: Favorites
12: Structure
13: I: Structure
14: Favorites
15: Structure
16: I: Structure
17: Favorites
18: Structure
19: I: Structure
20: Favorites
21: Structure
22: I: Structure
23: Favorites
24: Structure
25: I: Structure
26: Favorites
27: Structure
28: I: Structure
29: Favorites
30: Structure
31: I: Structure
```

public class ChainedRule implements ClaimingRule {
 private ClaimingRule head;
 private ClaimingRule tail;

 public ChainedRule(ClaimingRule head, ClaimingRule tail) {
 this.head = head;
 this.tail = tail;
 }

 @Override
 public Optional<Action> applicableTo(DeviceStatus status) {
 return this.head
 .applicableTo(status)
 .map(Optional::of)
 .orElse(tail.applicableTo(status));
 }
}

ChainedRule

head

tail

Find Debug TODO Terminal Messages Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > ChainedRule

Main

LifetimeWarranty.java VoidWarranty.java Article.java Demo.java ChainedRule.java StatusEqualityRule.java Action.java ClaimingRule.java

Project

Ant Build

m Maven

```
1: Project
2: Favorites
3: I: Structure
4: 
5:     public class ChainedRule implements ClaimingRule {
6:         private ClaimingRule head;
7:         private ClaimingRule tail;
8:
9:         public ChainedRule(ClaimingRule head, ClaimingRule tail) {
10:             this.head = head;
11:             this.tail = tail;
12:         }
13:
14:         @Override
15:         public Optional<Action> applicableTo(DeviceStatus status) {
16:             return this.head
17:                 .applicableTo(status)
18:                 .map(Optional::of)
19:                 .orElse(tail.applicableTo(status));
20:         }
21:     }
22:
23:     ChainedRule
24:     |
25:     +--> (Head Rule)
26:     |
27:     +--> (Tail Rule)
28:
29:
30:
31:
```

ChainedRule > applicableTo()

Find Debug TODO Terminal Messages Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > ChainedRule

Main

LifetimeWarranty.java VoidWarranty.java Article.java Demo.java ChainedRule.java StatusEqualityRule.java Action.java ClaimingRule.java

Project

Ant Build

m Maven

```
1: Project
2: Favorites
3: Structure
4: I: Structure
5: Favorites
6: Structure
7: I: Structure
8: Favorites
9: Structure
10: I: Structure
11: Favorites
12: Structure
13: I: Structure
14: Favorites
15: Structure
16: I: Structure
17: Favorites
18: Structure
19: I: Structure
20: Favorites
21: Structure
22: I: Structure
23: Favorites
24: Structure
25: I: Structure
26: Favorites
27: Structure
28: I: Structure
29: Favorites
30: Structure
31: I: Structure
```

public class ChainedRule implements ClaimingRule {
 private ClaimingRule head;
 private ClaimingRule tail;

 public ChainedRule(ClaimingRule head, ClaimingRule tail) {
 this.head = head;
 this.tail = tail;
 }

 @Override
 public Optional<Action> applicableTo(DeviceStatus status) {
 return this.head
 .applicableTo(status)
 .map(Optional::of)
 .orElse(tail.applicableTo(status));
 }
}

ChainedRule

Diagram illustrating the ChainedRule class structure:

```
graph LR; CR[ChainedRule] --> H[Head]; CR --> T[Tail]; H --> S1[StatusEqualityRule]; S1 --> A1[Action]; T --> S2[StatusEqualityRule]; S2 --> A2[Action];
```

Find Debug TODO Terminal Messages Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > ChainedRule

Main

LifetimeWarranty.java VoidWarranty.java Article.java Demo.java ChainedRule.java StatusEqualityRule.java Action.java ClaimingRule.java

Project

Ant Build

m Maven

```
1: Project
2: Favorites
3: Structure
4: I: Structure
5: 1: Project
6: 2: Favorites
7: 3: Structure
8: 4: I: Structure
9: 5: Structure
10: 6: Structure
11: 7: Structure
12: 8: Structure
13: 9: Structure
14: 10: Structure
15: 11: Structure
16: 12: Structure
17: 13: Structure
18: 14: Structure
19: 15: Structure
20: 16: Structure
21: 17: Structure
22: 18: Structure
23: 19: Structure
24: 20: Structure
25: 21: Structure
26: 22: Structure
27: 23: Structure
28: 24: Structure
29: 25: Structure
30: 26: Structure
31: 27: Structure
32: 28: Structure
33: 29: Structure
34: 30: Structure
35: 31: Structure
```

public class ChainedRule implements ClaimingRule {
 private ClaimingRule head;
 private ClaimingRule tail;

 public ChainedRule(ClaimingRule head, ClaimingRule tail) {
 this.head = head;
 this.tail = tail;
 }

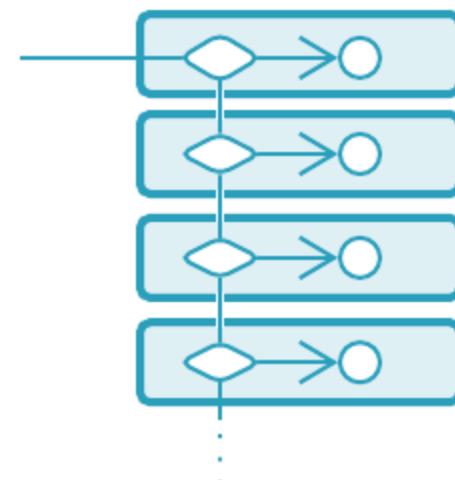
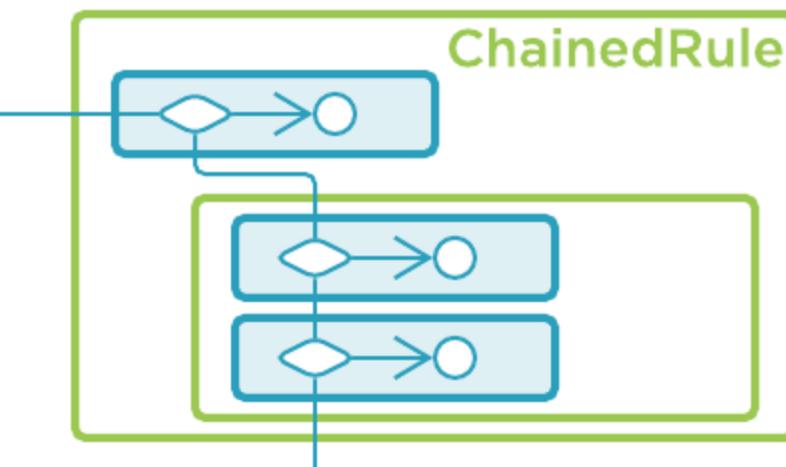
 @Override
 public Optional<Action> applicableTo(DeviceStatus status) {
 return this.head
 .applicableTo(status)
 .map(Optional::of)
 .orElse(tail.applicableTo(status));
 }
}

ChainedRule

ChainedRule > applicableTo()

Find Debug TODO Terminal Messages Run Event Log

```
4  
5     public class ChainedRule implements ClaimingRule {  
6         private ClaimingRule head;  
7         private ClaimingRule tail;  
8  
9         public ChainedRule(ClaimingRule head, ClaimingRule tail) {  
10            this.head = head;  
11            this.tail = tail;  
12        }  
13  
14        @Override  
15        public Optional<Action> applicableTo(DeviceStatus status) {  
16            return this.head  
17                .applicableTo(status)  
18                .map(Optional::of)  
19                .orElse(tail.applicableTo(status));  
20        }  
21    }
```



ChainedRule > applicableTo()

Pluralsight > src > com > codinghelmet > moreoojava > states > SensorFailedStatus

Main

Project

Pluralsight [Code] C:\Pluralsight

.idea

out

src

com.codinghelmet.moreoojava

rules

states

DeviceStatus

OperationalStatus

SensorFailedStatus

warranties

Action

Article

ClaimingRule

Demo

Main

Part

Warranty

Code.iml

External Libraries

Scratches and Consoles

1: Project

2: Favorites

3: Find

4: Debug

5: TODO

6: Terminal

7: Messages

8: Run

9: Event Log

Ant Build

Maven

```
4
5     public class SensorFailedStatus extends DeviceStatus {
6         private LocalDate failureDetectedOn;
7
8         @
9         public SensorFailedStatus(LocalDate detectedOn, OperationalStatus inState) {
10            super(inState.andSensorFailed());
11            this.failureDetectedOn = detectedOn;
12        }
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

sensor failed

Any state

New state

```
graph LR; A[Any state] -- "sensor failed" --> B[New state]
```

SensorFailedStatus

Pluralsight > src > com > codinghelmet > moreoojava > states > SensorFailedStatus

Main

Project

Pluralsight [Code] C:\Pluralsight

.idea

out

src

com.codinghelmet.moreoojava

rules

states

DeviceStatus

OperationalStatus

SensorFailedStatus

warranties

Action

Article

ClaimingRule

Demo

Main

Part

Warranty

Code.iml

External Libraries

Scratches and Consoles

1: Project

2: Favorites

3: Find

4: Debug

5: TODO

6: Terminal

7: Messages

8: Run

9: Event Log

Ant Build

Maven

```
4
5     public class SensorFailedStatus extends DeviceStatus {
6         private LocalDate failureDetectedOn;
7
8         @
9         public SensorFailedStatus(LocalDate detectedOn, OperationalStatus inState) {
10            super(inState.andSensorFailed());
11            this.failureDetectedOn = detectedOn;
12        }
13
14        public LocalDate getFailureDetectionDate() {
15            return this.failureDetectedOn;
16        }
17
18    }
19
20
21
22
23
24
25
26
27
28
29
30
31
```

SensorFailedStatus

```
classDiagram
    class AnyState
    class DeviceStatus
    class SensorFailedStatus {
        "LocalDate failureDetectedOn"
        "OperationalStatus state"
    }

    AnyState --> SensorFailedStatus : sensor failed
    Note over SensorFailedStatus: Includes sensor failed indicator
```

DeviceStatus

Any state

sensor failed

Includes sensor failed indicator

Project

- Pluralsight [Code] C:\Pluralsight
 - .idea
 - out
 - src
 - com.codinghelmet.moreoojava
 - rules
 - AppendingCondition
 - ChainedRule
 - ExtendingCondition
 - OperationalCondition
 - ReducedRule
 - RootCondition
 - RuleFixture
 - State
 - StatusTypeCondition
 - states
 - warranties
 - Action
 - Article
 - ClaimingRule
 - Demo
 - Main
 - Part
 - Warranty

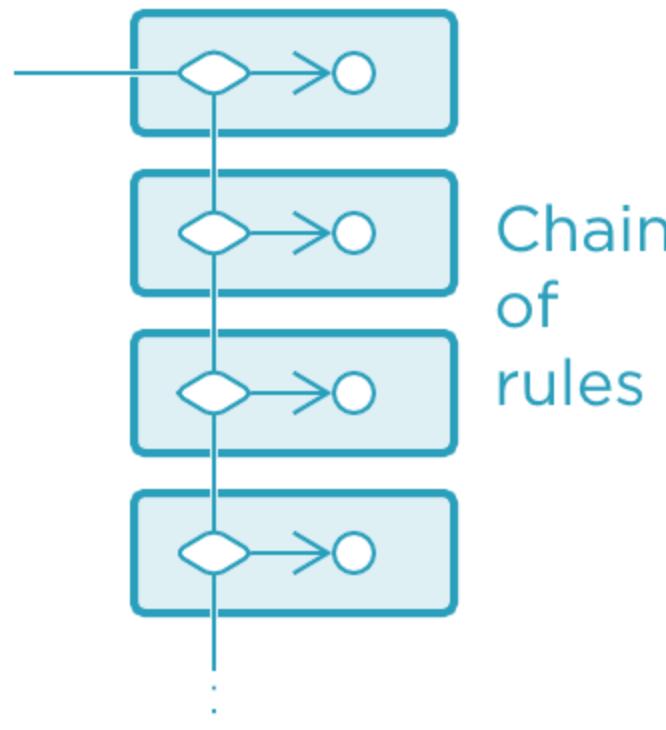
Code.iml

External Libraries

Scratches and Consoles

```
7
8     public interface ClaimingRule {
9         Optional<Action> applicableTo(DeviceStatus status);
10
11     default ClaimingRule orElse(ClaimingRule next) {
12         return new ChainedRule( head: this, next);
13     }
14 }
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
```

Chain
of
rules



Chain of rules

Pluralsight > src > com > codinghelmet > moreoojava > rules

Main

Project

1: Project

Pluralsight [Code] C:\Pluralsight

.idea

out

src

com.codinghelmet.moreoojava

rules

- AppendingCondition
- ChainedRule
- ExtendingCondition
- OperationalCondition
- ReducedRule
- RootCondition
- RuleFixture
- State
- StatusTypeCondition

states

warranties

- Action
- Article
- ClaimingRule
- Demo
- Main
- Part
- Warranty

Code.iml

External Libraries

Scratches and Consoles

2: Favorites

3: Find 5: Debug 6: TODO 7: Terminal 8: Messages 9: Run 10: Event Log

7

8 public interface ClaimingRule {

9 Optional<Action> applicableTo(DeviceStatus status);

10

11 default ClaimingRule orElse(ClaimingRule next) {

12 return new ChainedRule(head: this, next);

13 }

14 }

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

ClaimingRule

Chain of conditions

Pluralsight > src > com > codinghelmet > moreoojava > rules > RootCondition

Main

Project

1: Project

Pluralsight [Code] C:\Pluralsight

.idea

out

src

com.codinghelmet.moreoojava

rules

- AppendingCondition
- ChainedRule
- ExtendingCondition
- OperationalCondition
- ReducedRule
- RootCondition
- RuleFixture
- State
- StatusTypeCondition

states

warranties

- Action
- Article
- ClaimingRule
- Demo
- Main
- Part
- Warranty

Code.iml

External Libraries

Scratches and Consoles

2: Favorites

3: Find 5: Debug 6: TODO 7: Terminal 8: Messages 9: Run 10: Event Log

11: RootCondition

```
public interface RootCondition<T extends DeviceStatus> {
    Optional<T> applicableTo(DeviceStatus status);

    default ClaimingRule applies(Consumer<T> action) {
        return new RuleFixture<>( condition: this, action);
    }
}
```

DeviceStatus

DeviceStatus

T

RootCondition<T>

Diagram illustrating the relationship between DeviceStatus and T:

- A green rounded rectangle labeled "DeviceStatus" is at the top.
- An oval labeled "DeviceStatus" is in the middle-left.
- An oval labeled "T" is in the middle-right.
- A blue arrow points from the "DeviceStatus" oval to the "T" oval.
- A blue arrow points from the "T" oval up to the "DeviceStatus" rectangle.

Pluralsight > src > com > codinghelmet > moreoojava > rules > ExtendingCondition

Project 1: Pluralsight [Code] C:\Pluralsight

Demo.java DeviceStatus.java SensorFailedStatus.java Article.java ClaimingRule.java RootCondition.java ExtendingCondition.java

6
7 public interface ExtendingCondition<T1 extends DeviceStatus, T2 extends DeviceStatus> {
8 Optional<T2> applicableTo(T1 status);
9 }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

ExtendingCondition

Diagram illustrating the class hierarchy:

```
graph TD; DeviceStatus --> T; T --> U; U --> V; RootCondition<T> --> T; ExtendingCondition<T, U> --> U; ExtendingCondition<U, V> --> V;
```

The diagram shows the relationship between the `DeviceStatus` interface at the top, which has a generalization arrow pointing to a class `T`. Class `T` has generalization arrows pointing to classes `U` and `V`. Below the class hierarchy, there are two interfaces: `RootCondition<T>` and two `ExtendingCondition` interfaces, `ExtendingCondition<T, U>` and `ExtendingCondition<U, V>`. Arrows point from each of these interfaces to their respective class instances (`T`, `U`, and `V`) in the hierarchy.

Pluralsight > src > com > codinghelmet > moreoojava > rules > AppendingCondition

Project Main Find Debug TODO Terminal Messages Run Event Log

```
public class AppendingCondition<T extends DeviceStatus, T1 extends DeviceStatus>
    implements RootCondition<T1> {
    private RootCondition<T> first;
    private ExtendingCondition<T, T1> second;

    public AppendingCondition(RootCondition<T> first,
                             ExtendingCondition<T, T1> second) {
        this.first = first;
        this.second = second;
    }

    @Override
    public Optional<T1> applicableTo(DeviceStatus status) {
        return this.first.applicableTo(status)
            .map(this.second::applicableTo)
            .orElse(Optional.empty());
    }
}
```

Diagram illustrating the composition of conditions:

```
graph LR
    DS1((DeviceStatus)) --> T1_1((T1))
    T1_1 --> T2_1((T2))
    DS2((DeviceStatus)) --> T1_2((T2))
    T1_2 --> T2_2((T2))
```

Pluralsight > src > com > codinghelmet > moreoojava > rules > AppendingCondition

Project Main Find Debug TODO Terminal Messages Run Event Log

```
public class AppendingCondition<T extends DeviceStatus, T1 extends DeviceStatus>
    implements RootCondition<T1> {
    private RootCondition<T> first;
    private ExtendingCondition<T, T1> second;

    public AppendingCondition(RootCondition<T> first,
                             ExtendingCondition<T, T1> second) {
        this.first = first;
        this.second = second;
    }

    @Override
    public Optional<T1> applicableTo(DeviceStatus status) {
        return this.first.applicableTo(status)
            .map(this.second::applicableTo)
            .orElse(Optional.empty());
    }
}
```

Diagram illustrating the composition of conditions:

```
graph LR
    DS1((DeviceStatus)) --> T1((T1))
    DS2((DeviceStatus)) --> T2((T2))
    DS1 --- DS2
    T1 --- T2
    DS1 --- T2
```

The diagram shows two `DeviceStatus` objects (one gray, one orange). The gray `DeviceStatus` has arrows pointing to `T1` and `T2`. The orange `DeviceStatus` also has arrows pointing to `T1` and `T2`. A red arrow points from the word "RootCondition" in the code to the orange `DeviceStatus` object.

Pluralsight > src > com > codinghelmet > moreoojava > rules > RootCondition

Main

Project

Pluralsight [Code] C:\Pluralsight

.idea

out

src

com.codinghelmet.moreoojava

rules

- AppendingCondition
- ChainedRule
- ExtendingCondition
- OperationalCondition
- ReducedRule
- RootCondition
- RuleFixture
- State
- StatusTypeCondition

states

warranties

- Action
- Article
- ClaimingRule
- Demo
- Main
- Part
- Warranty

Code.iml

External Libraries

Scratches and Consoles

1: Project

2: Favorites

3: Find 5: Debug 6: TODO 7: Terminal 8: Messages 9: Run 10: Event Log

11: Main 12: Ant Build 13: Maven

```
public interface RootCondition<T extends DeviceStatus> {  
    Optional<T> applicableTo(DeviceStatus status);  
  
    default ClaimingRule applies(Consumer<T> action) {  
        return new RuleFixture<>( condition: this, action);  
    }  
}
```

RootCondition

The diagram illustrates the relationship between the **RootCondition<T>** and **RuleFixture<T>** classes. The **RootCondition<T>** class is shown in a red-bordered box with a red arrow pointing from its **DeviceStatus** association to its **T** generic type. A large grey arrow points from the **RootCondition<T>** box to the **RuleFixture<T>** class, which is shown in a blue-bordered box.

Pluralsight > src > com > codinghelmet > moreoojava > rules > RuleFixture

Project Main Find Debug TODO Terminal Messages Run Event Log

1: Project

Pluralsight [Code] C:\Pluralsight

- .idea
- out
- src
 - com.codinghelmet.moreoojava
 - AppendingCondition
 - ChainedRule
 - ExtendingCondition
 - OperationalCondition
 - ReducedRule
 - RootCondition
 - RuleFixture
 - State
 - StatusTypeCondition
 - states
 - warranties
 - Action
 - Article
 - ClaimingRule
 - Demo
 - Main
 - Part
 - Warranty
- Code.iml
- External Libraries
- Scratches and Consoles

2: Favorites

Ant Build Maven

```
public class RuleFixture<T extends DeviceStatus> implements ClaimingRule {  
    private RootCondition<T> condition;  
    private Consumer<T> action;  
  
    public RuleFixture(RootCondition<T> condition, Consumer<T> action) {  
        this.condition = condition;  
        this.action = action;  
    }  
  
    @Override  
    public Optional<Action> applicableTo(DeviceStatus status) {  
        return this.condition  
            .applicableTo(status)  
            .map(s -> new ReducedRule<>(s, this.action));  
    }  
}
```

Abstract

Concrete

Diagram illustrating the relationship between **RootCondition<T>**, **RuleFixture<T>**, and **ClaimingRule**:

```
graph TD; RootCondition["RootCondition<T>"] --> RuleFixture["RuleFixture<T>"]; RuleFixture --> ClaimingRule["ClaimingRule"]
```

The diagram shows a flow from the **RootCondition<T>** box to the **RuleFixture<T>** box, and then from the **RuleFixture<T>** box to the **ClaimingRule** box.

1: Project

2: Favorites

Ant Build Maven

Find Debug TODO Terminal Messages Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > rules > ReducedRule

Project Main Find Debug TODO Terminal Messages Run Event Log

1: Project 2: Favorites 3: Structure 4: Favorites

Condition.java RuleFixture.java ReducedRule.java ExtendingCondition.java OperationalCondition.java StatusTypeCondition.java

7
8 public class ReducedRule<T extends DeviceStatus> implements Action {
9 private T status;
10 private Consumer<T> action;
11
12 public ReducedRule(T status, Consumer<T> action) {
13 this.status = status;
14 this.action = action;
15 }
16
17 @Override
18 public void apply() {
19 this.action.accept(this.status);
20 }
21
22 }
23
24
25
26
27
28
29
30
31
32
33
34

ReducedRule

```
graph TD; RootCondition["RootCondition<T>"] -- "Red Arrow" --> Abstract[Abstract]; RuleFixture["RuleFixture<T>"] -- "Blue Arrow" --> Concrete[Concrete]; ClaimingRule["ClaimingRule"]; Action["Action"]
```

Pluralsight > src > com > codinghelmet > moreoojava > rules > ExhaustiveRulesBuilder

Main

Project

Demo.java X

ExhaustiveRulesBuilder.java X

ClaimingRulesBuilder.java X

Ant Build

Maven

1: Project

2: Favorites

3: I: Structure

4: Favorites

```
45
46     private ClaimingRule allFineRule() {
47         return State.matching(
48             OperationalStatus.allFine()).applies(
49             this::moneyBack);
50     }
51
52     private ClaimingRule notOperationalRule() {
53         return State.matching(
54             OperationalStatus.notOperational()).applies(
55             s -> {
56                 this.moneyBack(s);
57                 this.express(s);
58             });
59     }
60
61     private ClaimingRule visiblyDamagedRule() {
62         return State.matching(
63             OperationalStatus.visiblyDamaged()).applies(
64             this::doNothing);
65     }
66
67     private ClaimingRule sensorFailedRule() {
68         return State.matching(
69             OperationalStatus.sensorFailed(),
70             SensorFailedStatus.class).applies(
71             s -> {
72                 this.moneyBack(s);
```

When state

ExhaustiveRulesBuilder

TODO Terminal Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > rules > ExhaustiveRulesBuilder

Main

Project

Demo.java

ExhaustiveRulesBuilder.java

ClaimingRulesBuilder.java

Ant Build

Maven

1: Project

2: Favorites

3: I: Structure

4: Favorites

```
45
46     private ClaimingRule allFineRule() {
47         return State.matching(
48             OperationalStatus.allFine()).applies(
49                 this::moneyBack);
50     }
51
52     private ClaimingRule notOperationalRule() {
53         return State.matching(
54             OperationalStatus.notOperational()).applies(
55                 s -> {
56                     this.moneyBack(s);
57                     this.express(s);
58                 });
59     }
60
61     private ClaimingRule visiblyDamagedRule() {
62         return State.matching(
63             OperationalStatus.visiblyDamaged()).applies(
64                 this::doNothing);
65     }
66
67     private ClaimingRule sensorFailedRule() {
68         return State.matching(
69             OperationalStatus.sensorFailed(),
70             SensorFailedStatus.class).applies(
71                 s -> {
72                     this.moneyBack(s);
```

When state is matching

ExhaustiveRulesBuilder

TODO Terminal Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > rules > ExhaustiveRulesBuilder

Main

Project

Demo.java

ExhaustiveRulesBuilder.java

ClaimingRulesBuilder.java

Ant Build

Maven

1: Project

2: Favorites

3: I: Structure

4: Favorites

```
45
46     private ClaimingRule allFineRule() {
47         return State.matching(
48             OperationalStatus.allFine()).applies(
49                 this::moneyBack);
50     }
51
52     private ClaimingRule notOperationalRule() {
53         return State.matching(
54             OperationalStatus.notOperational()).applies(
55                 s -> {
56                     this.moneyBack(s);
57                     this.express(s);
58                 });
59     }
60
61     private ClaimingRule visiblyDamagedRule() {
62         return State.matching(
63             OperationalStatus.visiblyDamaged()).applies(
64                 this::doNothing);
65     }
66
67     private ClaimingRule sensorFailedRule() {
68         return State.matching(
69             OperationalStatus.sensorFailed(),
70             SensorFailedStatus.class).applies(
71                 s -> {
72                     this.moneyBack(s);
```

When state is matching operational status

ExhaustiveRulesBuilder

TODO Terminal Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > rules > ExhaustiveRulesBuilder

Main

Project

Demo.java

ExhaustiveRulesBuilder.java

ClaimingRulesBuilder.java

Ant Build

Maven

1: Project

2: Favorites

3: I: Structure

4: Favorites

```
45
46     private ClaimingRule allFineRule() {
47         return State.matching(
48             OperationalStatus.allFine()).applies(
49                 this::moneyBack);
50     }
51
52     private ClaimingRule notOperationalRule() {
53         return State.matching(
54             OperationalStatus.notOperational()).applies(
55                 s -> {
56                     this.moneyBack(s);
57                     this.express(s);
58                 });
59     }
60
61     private ClaimingRule visiblyDamagedRule() {
62         return State.matching(
63             OperationalStatus.visiblyDamaged()).applies(
64                 this::doNothing);
65     }
66
67     private ClaimingRule sensorFailedRule() {
68         return State.matching(
69             OperationalStatus.sensorFailed(),
70             SensorFailedStatus.class).applies(
71                 s -> {
72                     this.moneyBack(s);
```

When state is matching operational status "not operational"

ExhaustiveRulesBuilder

TODO Terminal Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > rules > ExhaustiveRulesBuilder

Main

Project

Demo.java

ExhaustiveRulesBuilder.java

ClaimingRulesBuilder.java

Ant Build

Maven

```
1: Project
2: Favorites
3: I: Structure
4: 2: Favorites
5: 1: Project
6: Demo.java
7: ExhaustiveRulesBuilder.java
8: ClaimingRulesBuilder.java
9: Ant Build
10: Maven
11: 45
12: 46     private ClaimingRule allFineRule() {
13:         return State.matching(
14:             OperationalStatus.allFine()).applies(
15:                 this::moneyBack);
16:     }
17:
18: 47
19: 48
20: 49     private ClaimingRule notOperationalRule() {
21:         return State.matching(
22:             OperationalStatus.notOperational()).applies(
23:                 s -> {
24:                     this.moneyBack(s);
25:                     this.express(s);
26:                 });
27:     }
28:
29: 50
30: 51
31: 52
32: 53     private ClaimingRule visiblyDamagedRule() {
33:         return State.matching(
34:             OperationalStatus.visiblyDamaged()).applies(
35:                 this::doNothing);
36:     }
37:
38: 59
39: 60
40: 61
41: 62     private ClaimingRule sensorFailedRule() {
42:         return State.matching(
43:             OperationalStatus.sensorFailed(),
44:             SensorFailedStatus.class).applies(
45:                 s -> {
46:                     this.moneyBack(s);
47:                 });
48:     }
49:
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
```

ExhaustiveRulesBuilder

When state is matching operational status "not operational" it applies

Pluralsight > src > com > codinghelmet > moreoojava > rules > ExhaustiveRulesBuilder

Main

Project

Demo.java

ExhaustiveRulesBuilder.java

ClaimingRulesBuilder.java

Ant Build

Maven

1: Project

2: Favorites

3: I: Structure

4: Favorites

```
45
46     private ClaimingRule allFineRule() {
47         return State.matching(
48             OperationalStatus.allFine()).applies(
49                 this::moneyBack);
50     }
51
52     private ClaimingRule notOperationalRule() {
53         return State.matching(
54             OperationalStatus.notOperational()).applies(
55                 s -> {
56                     this.moneyBack(s),
57                     this.express(s);
58                 });
59     }
60
61     private ClaimingRule visiblyDamagedRule() {
62         return State.matching(
63             OperationalStatus.visiblyDamaged()).applies(
64                 this::doNothing);
65     }
66
67     private ClaimingRule sensorFailedRule() {
68         return State.matching(
69             OperationalStatus.sensorFailed(),
70             SensorFailedStatus.class).applies(
71                 s -> {
72                     this.moneyBack(s);
```

When state is matching operational status "not operational" it applies money back

ExhaustiveRulesBuilder

TODO Terminal Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > rules > ExhaustiveRulesBuilder

Main

Project

Demo.java

ExhaustiveRulesBuilder.java

ClaimingRulesBuilder.java

Ant Build

Maven

1: Project

2: Favorites

3: I: Structure

4: Favorites

```
45
46     private ClaimingRule allFineRule() {
47         return State.matching(
48             OperationalStatus.allFine()).applies(
49                 this::moneyBack);
50     }
51
52     private ClaimingRule notOperationalRule() {
53         return State.matching(
54             OperationalStatus.notOperational()).applies(
55                 s -> {
56                     this.moneyBack(s);
57                     this.express();
58                 });
59     }
60
61     private ClaimingRule visiblyDamagedRule() {
62         return State.matching(
63             OperationalStatus.visiblyDamaged()).applies(
64                 this::doNothing);
65     }
66
67     private ClaimingRule sensorFailedRule() {
68         return State.matching(
69             OperationalStatus.sensorFailed(),
70             SensorFailedStatus.class).applies(
71                 s -> {
72                     this.moneyBack(s);
```

When state is matching operational status "not operational" it applies money back and express claim

ExhaustiveRulesBuilder

TODO Terminal Run Event Log

Pluralsight > src > com > codinghelmet > moreoojava > rules > ExhaustiveRulesBuilder

Main

Project

Demo.java

ExhaustiveRulesBuilder.java

ClaimingRulesBuilder.java

Ant Build

Maven

```
1: Project
2: Favorites
3: I: Structure
4: 2: Favorites
5: 1: Project
6: 2: Favorites
7: 3: I: Structure
8: 4: Run
9: 5: Event Log
```

45

46 private ClaimingRule allFineRule() {

47 return State.matching(

48 OperationalStatus.allFine()).applies(

49 this::moneyBack);

50 }

51

52 private ClaimingRule notOperationalRule() {

53 return State.matching(

54 OperationalStatus.notOperational()).applies(

55 s -> {

56 this.moneyBack(s);

57 this.express(s);

58 });

59 }

60

61 private ClaimingRule visiblyDamagedRule() {

62 return State.matching(

63 OperationalStatus.visiblyDamaged()).applies(

64 this::doNothing);

65 }

66

67 private ClaimingRule sensorFailedRule() {

68 return State.matching(

69 OperationalStatus.sensorFailed(),

70 SensorFailedStatus.class).applies(

71 s -> {

72 this.moneyBack(s);

ExhaustiveRulesBuilder

When state is matching operational status "not operational" it applies money back and express claim

Summary



Why decomposing a chain of if-elses?

- Control flow is rigid
- Doesn't support changing requests
- Implemented at the consuming end



Summary



Introducing rules instead of if-elses

- Boolean condition + action are a rule
- A rule is either applicable or not
- Multiple rules form a chain
- Control is passed down the chain
- Rules can also be independent
- Rule can test multiple conditions
- Complex rules can be constructed



Summary



Surviving the complexity of rules

- Use rule builders to contain complexity
- Concrete builders implement concrete views on business
- Rule builders can be substituted



Summary



Managing a complex business domain

- State pattern to model discrete state
- Polymorphic calls to substitute Boolean tests
- Filtering interface and optional objects to model object existence
- Composite pattern to manage multitudes of objects
- Strategy pattern to vary behavior
- Rules pattern to model business rules
- Domain-specific Language to make the code composable



Summary



Next module:
Refactoring to Bind it All Together

