

Staying Focused on Domain Logic with Streams



Zoran Horvat
CEO AT CODING HELMET

@zoranh75 <http://codinghelmet.com>



Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java x Demo.java x Main.java x

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce()

11 .get();

12 }

13

14 public void run() {

15 }

16 }

17

18

19

20

21 Stream ACC

22

23

24

25

26

27

28

29

30

31

32

Demo > findCheapest1()

Event Log

The diagram illustrates the reduce operation in a Java Stream. It shows a Stream of five light blue squares above a green ACC (Accumulator) square. An arrow points from the Stream to the ACC, indicating the flow of data through the stream to the accumulator.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

Project

1: Project

Ant Build

m Maven

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) ->)

11 .get();

12 }

13

14 public void run() {

15 }

16 }

17

18

19

20 Stream ACC f

21

22

23

24

25

26

27

28

29

30

31

32

Demo > findCheapest1()

Event Log

2: Favorites

3: I: Structure

4: TODO Terminal

5: Main

6: Event Log

The diagram illustrates the Java Stream reduce operation. It shows a Stream of five light blue squares, labeled "Stream". An arrow points from the Stream to a green square labeled "ACC", representing the initial value of the accumulator. Another arrow points from the Stream to a red circle labeled "f", representing the accumulator function. The Stream consists of five light blue squares.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

1: Project

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) ->)

11 .get();

12 }

13

14 public void run() {

15 }

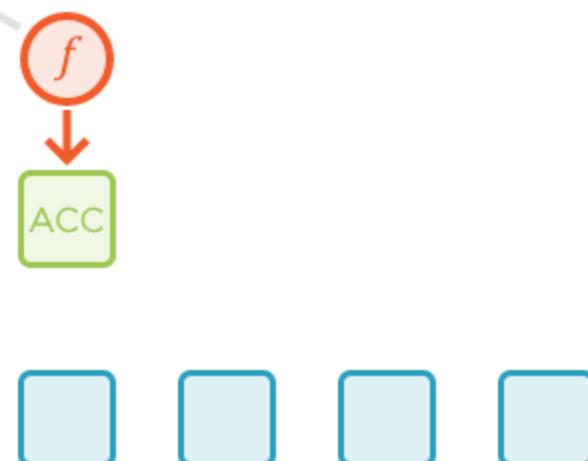
16 }

17

18

19

20

21 Stream 

22

23

24

25

26

27

28

29

30

31

32

2: Favorites

2: I: Structure

2: Favorites

Demo > findCheapest1()

Event Log

Ant Build

m Maven

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 public void run() {

16 }

17 }

18

19

20 Stream ACC f

21

22

23

24

25

26

27

28

29

30

31

32

Demo > findCheapest1()

Event Log

Ant Build

m Maven

```
graph TD; Stream[Stream] --> ACC[ACC]; ACC --> f((f))
```

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

1: Project

2: Favorites

3: I: Structure

4: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 public void run() {

16 }

17 }

18

19

20 Stream 

21

22

23

24

25

26

27

28

29

30

31

32

Demo > findCheapest10

Event Log

Ant Build

m Maven

6 TODO Terminal

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 public void run() {

16 }

17 }

18

19

20 Stream ACC f

21

22

23

24

25

26

27

28

29

30

31

32

Demo > findCheapest1()

Event Log

Ant Build

m Maven

```
graph TD; Stream[Stream] --> ACC[ACC]; Stream --> f((f)); Stream --> ACC; Stream --> f; Stream --> ACC; Stream --> f; Stream --> ACC;
```

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

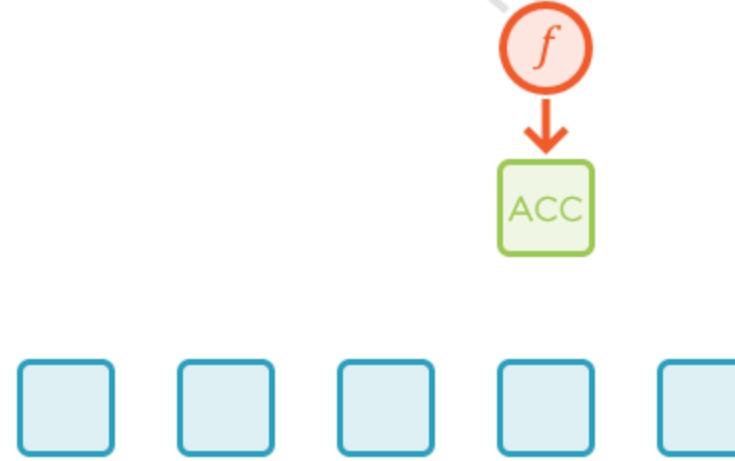
15 public void run() {

16 }

17 }

18

19

20 Stream 

21

22

23

24

25

26

27

28

29

30

31

32

Demo > findCheapest1()

Event Log

Ant Build

m Maven

6 TODO Terminal

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 public void run() {

16 }

17 }

18

19

20 Stream ACC f

21

22

23

24

25

26

27

28

29

30

31

32

Demo > findCheapest1()

Event Log

Ant Build

m Maven

The diagram illustrates the state transition of a Stream. It shows a green arrow pointing from a light blue box labeled "ACC" to a red circle labeled "f". Below the "ACC" box are five other light blue boxes, representing the elements of the stream. This visualizes how the accumulator (ACC) is updated with each element (represented by the blue boxes) to produce the final result (f).

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

Painter.java X Demo.java X Main.java X

5

6 public class Demo {

7 @ ...

8 private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

9 return painters.stream()

10 .filter(Painter::isAvailable)

11 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

12 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

13 .get();

14 }

15 public void run() {

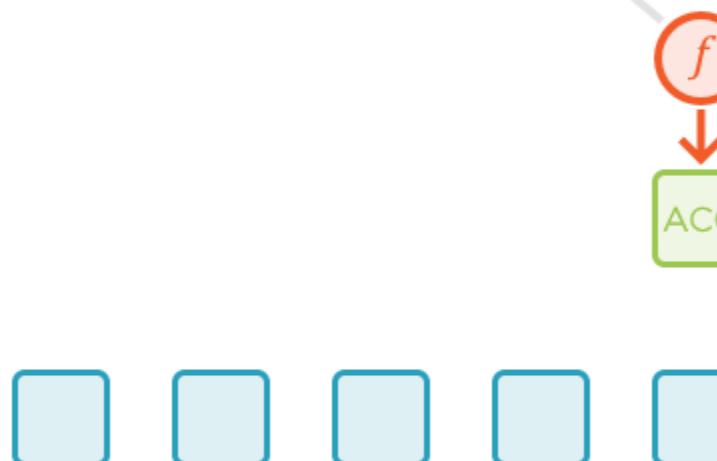
16 }

17 }

18

19

20

21 Stream 

22

23

24

25

26

27

28

29

30

31

32

2: Favorites

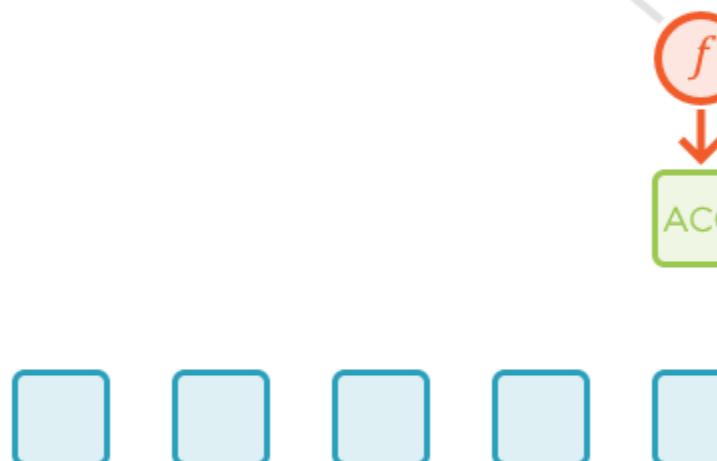
3: I: Structure

Demo > findCheapest1()

6 TODO Terminal Event Log

Ant Build

m Maven



The diagram illustrates the execution flow of a Java Stream reduce operation. It starts with a Stream of five light blue squares, each representing an element from the input list. An arrow points from this Stream to a green square labeled 'ACC' (Accumulator). Inside the 'ACC' square is a red circle containing the letter 'f', representing the identity function used in the reduction. The code snippet above shows the corresponding Java code for this operation.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 public void run() {

16 }

17 }

18

19

20

21 Stream

22

23

24

25

26

27

28

29

30

31

32

Demo > findCheapest1()

Event Log

Ant Build

m Maven

result

ACC

The diagram illustrates the execution flow of a Java Stream reduce operation. The code shows a stream being filtered and then reduced using the `.reduce` method. The `reduce` method takes a function that takes an accumulator (acc) and a current element (current), and returns a new accumulator. The `compareTo` method is used to compare the compensation values. The result of the reduction is stored in the variable `ACC`. The word "Stream" is followed by five empty blue boxes, likely representing the elements of the stream being processed.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

Project

1: Project

Ant Build

m Maven

2: Favorites

1: I-Structure

2: I-Structure

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO,

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream seed

29 ACC

30

31

32

Demo > getTotalCost()

Event Log

TODO Terminal

The diagram illustrates the reduce operation in Java Streams. It shows a Stream of five light blue squares. An arrow labeled "seed" points from the first square to a green hexagon labeled "ACC" (Accumulator).

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

Project

1: Project

2: Favorites

3: I: Structure

4: 1: Project

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO, (acc, painter) ->)

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream

29

30

31

32

Demo > getTotalCost()

Event Log

Ant Build

m Maven

The diagram illustrates the reduce operation in Java Streams. It shows a Stream of five light blue squares being reduced into a single orange circle labeled 'f'. This circle is labeled 'f' and has a green arrow pointing to it from a green hexagon labeled 'ACC'.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO, (acc, painter) ->)

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream f ACC

29

30

31

32

Demo > getTotalCost()

Event Log

The screenshot shows a Java code editor with a focus on a class named 'Demo'. The code contains two static methods: 'findCheapest1' and 'getTotalCost'. The 'getTotalCost' method uses a Java Stream to calculate the total cost based on a list of 'Painter' objects. A callout diagram highlights the 'reduce' operation, showing a red circle labeled 'f' representing the function being applied to each painter, and a green hexagon labeled 'ACC' representing the accumulator state. Below the diagram, five empty blue squares represent the stream elements. The code editor interface includes toolbars, a sidebar with project and build information, and a bottom navigation bar.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO, (acc, painter) ->)

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream

29 ACC

30 f

31

32

Demo > getTotalCost()

Event Log

The diagram illustrates the execution of a reduce operation on a stream. A Stream of five light blue squares (representing elements) is shown merging into a single red circle labeled 'f'. An arrow points from a green hexagon labeled 'ACC' (representing the accumulator) to the first square. Another arrow points from the 'f' circle to the last square, indicating the flow of data through the stream.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO, (acc, painter) ->)

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream

29

30

31

32

Demo > getTotalCost()

Event Log

Ant Build

m Maven

The screenshot shows a Java code editor with a focus on the `getTotalCost` method. The code uses a stream API to calculate the total cost of painting a given area. The `reduce` operation is highlighted with a yellow background. A diagram below the code illustrates the process: a red circle labeled 'f' with a downward arrow points to a green hexagon labeled 'ACC', representing the function being applied to each element and its accumulation into a single result.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

Project

1: Project

Ant Build

m Maven

2: Favorites

3: I: Structure

4: 1: Demo

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO, (acc, painter) ->)

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream ACC f

29

30

31

32

Demo > getTotalCost()

Event Log

6 TODO Terminal

The diagram illustrates the state transition of a Stream reduce operation. It shows a Stream of five light blue squares, an ACC (Accumulator) hexagon labeled 'ACC', and a function 'f' in a red circle. A green arrow points from the Stream to the ACC, and a blue arrow points from the ACC to the function 'f'.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO, (acc, painter) ->  

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream     

29

30

31

32

Demo > getTotalCost()

Event Log



Ant Build

m Maven

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X Main.java X

Project

1: Project

2: Favorites

3: I: Structure

4: 1: Demo

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO, (acc, painter) ->)

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream [] [] [] [] []

29

30

31

32

Demo > getTotalCost()

Event Log

Ant Build

m Maven

```
graph TD; ACC((ACC)) --> f((f)); f --> E1[ ]; f --> E2[ ]; f --> E3[ ]; f --> E4[ ]; f --> E5[ ]
```

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO, (acc, painter) ->)

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream [] [] [] [] []

29

30

31

32

Demo > getTotalCost()

Event Log

The screenshot shows an IDE interface with Java code in the main editor. The code defines two static methods: `findCheapest1` and `getTotalCost`. The `getTotalCost` method uses a stream to reduce a list of `Painter` objects into a `Money` object. A callout diagram highlights the `.reduce` operation, showing a red circle labeled 'f' with a downward arrow pointing to a green hexagon labeled 'ACC', representing the accumulator function. Below the stream, five blue squares are labeled 'Stream'.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO, (acc, painter) ->)

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream [] [] [] [] []

29

30

31

32

Demo > getTotalCost()

Event Log

The diagram illustrates the execution flow of a Java Stream reduce operation. It shows a sequence of five light blue squares representing the stream elements. An arrow points from the first square to a green hexagon labeled 'ACC' (Accumulator). Another arrow points from the 'ACC' hexagon to a red circle labeled 'f', representing the reduction function. This visualizes how the stream elements are aggregated through the accumulator.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

Painter.java X Demo.java X Main.java X

5

6 public class Demo {

7 @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {

8 return painters.stream()

9 .filter(Painter::isAvailable)

10 .reduce((acc, current) -> acc.estimateCompensation(sqMeters)

11 .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)

12 .get();

13 }

14

15 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

16 return painters.stream()

17 .filter(Painter::isAvailable)

18 .reduce(Money.ZERO, (acc, painter) ->)

19 }

20

21 public void run() {

22 }

23 }

24

25

26

27

28 Stream [] [] [] [] []

29

30

31

32

Demo > getTotalCost()

2: Favorites

1: Structure

Ant Build

m Maven

Event Log

The diagram illustrates the execution flow of a Java Stream reduce operation. It shows a red circle containing the letter 'f' with a downward arrow pointing to a green hexagon containing the letters 'ACC'. A grey arrow originates from the code line 'reduce(Money.ZERO, (acc, painter) ->)' and points to the 'f' circle, indicating the mapping function being applied to each element in the stream.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

Painter.java X Demo.java X Main.java X

Ant Build

m Maven

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

5
6     public class Demo {
7         @
8             private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
9                 return painters.stream()
10                .filter(Painter::isAvailable)
11                .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
12                      .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
13                .get();
14
15         @
16             private static Money getTotalCost(double sqMeters, List<Painter> painters) {
17                 return painters.stream()
18                    .filter(Painter::isAvailable)
19                    .reduce(Money.ZERO, (acc, painter) -> painter.estimateCompensation(sqMeters).add(acc),
20                           (cost1, cost2) -> cost1.add(cost2));
21
22         public void run() {
23
24     }
25
26
27
28     Stream
29
30
31
32
```

Demo > getTotalCost()

Event Log

The screenshot shows a Java code editor with a Stream diagram overlaid. The code is a demonstration of Java Streams. The `getTotalCost` method uses a stream to filter available painters and then reduce them into a total cost using a custom reduction function. A green box highlights the reduction part of the code. A green arrow points from the word 'result' to a green hexagon labeled 'ACC', which represents the accumulator in the reduction process. Below the Stream label, there are five blue squares representing the elements being processed.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java Demo.java PaintersStream.java Main.java

1: Project

6

7 public class Demo {

8 @ private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {

9 return painters.stream()

10 .filter(Painter::isAvailable)

11 .min(Comparator.comparing(painter -> painter.estimateCompensation(sqMeters))));

12 }

13

14 private static Optional<Painter> findCheapest2(double sqMeters, List<Painter> painters) {

15 return Painter.stream(painters).available().cheapest(sqMeters);

16 }

17

18 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

19 return painters.stream() Stream<Painter>

20 .filter(Painter::isAvailable) Stream<Painter>

21 .map(painter -> painter.estimateCompensation(sqMeters)) Stream<Money>

22 .reduce(Money::add) Optional<Money>

23 .orElse(Money.ZERO);

24 }

25

26 public void run() {

27 }

28 }

29

30

31

32

33

Demo > findCheapest2()

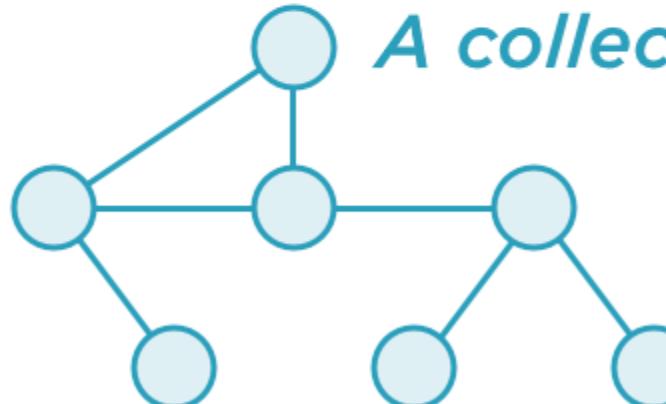
1: I Structure

2: Favorites

Ant Build

m Maven

A collection



Event Log

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X PaintersStream.java X Main.java X

1: Project

6

7 public class Demo {

8 @ private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {

9 return painters.stream()

10 .filter(Painter::isAvailable)

11 .min(Comparator.comparing(painter -> painter.estimateCompensation(sqMeters))));

12 }

13

14 private static Optional<Painter> findCheapest2(double sqMeters, List<Painter> painters) {

15 return Painter.stream(painters).available().cheapest(sqMeters);

16 }

17

18 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

19 return painters.stream() Stream<Painter>

20 .filter(Painter::isAvailable) Stream<Painter>

21 .map(painter -> painter.estimateCompensation(sqMeters)) Stream<Money>

22 .reduce(Money::add) Optional<Money>

23 .orElse(Money.ZERO);

24 }

25

26 public void run() {

27 }

28 }

29

30

31

32

33

Demo > findCheapest2()

Ant Build

m Maven

A collection



2: I: Structure

2: Favorites

Todo Terminal Event Log

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

Painter.java X Demo.java X PaintersStream.java X Main.java X

6

7 public class Demo {

8 private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {

9 return painters.stream()

10 .filter(Painter::isAvailable)

11 .min(Comparator.comparing(painter -> painter.estimateCompensation(sqMeters))));

12 }

13

14 private static Optional<Painter> findCheapest2(double sqMeters, List<Painter> painters) {

15 return Painter.stream(painters).available().cheapest(sqMeters);

16 }

17

18 private static Money getTotalCost(double sqMeters, List<Painter> painters) {

19 return painters.stream() Stream<Painter>

20 .filter(Painter::isAvailable) Stream<Painter>

21 .map(painter -> painter.estimateCompensation(sqMeters)) Stream<Money>

22 .reduce(Money::add) Optional<Money>

23 .orElse(Money.ZERO);

24 }

25

26 public void run() {

27 }

28 }

29

30

31

32

33

Demo > findCheapest2()

Event Log

Ant Build

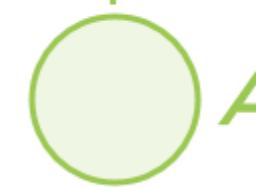
m Maven

A hash table?

A dictionary?

A list?

A collection



The image shows a Java code editor with several annotations and tool tips. Annotations include 'A hash table?' (yellow), 'A dictionary?' (yellow), 'A list?' (yellow), and 'A collection' (green). A large green circle highlights the word 'Collection'.

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X PaintersStream.java X Main.java X

1: Project

6

7 public class Demo {

8 @ private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {

9 return painters.stream()

10 .filter(Painter::isAvailable)

11 .min(Comparator.comparing(painter -> painter.estimateCompensation(sqMeters))));

12 }

13

14 private static Optional<Painter> findCheapest2(double sqMeters, List<Painter> painters) {

15 return Painter.stream(painters).available().cheapest(sqMeters);

16 }

17

18 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

19 return painters.stream() Stream<Painter>

20 .filter(Painter::isAvailable) Stream<Painter>

21 .map(painter -> painter.estimateCompensation(sqMeters)) Stream<Money>

22 .reduce(Money::add) Optional<Money>

23 .orElse(Money.ZERO);

24 }

25

26 public void run() {

27 }

28 }

29

30

31

32

33

Demo > findCheapest2()

1: Structure

2: Favorites

Ant Build

m Maven

A stream ✓

A collection

Event Log

6 TODO Terminal

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X PaintersStream.java X Main.java X

Project

1: Project

6

7 public class Demo {

8 @ private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {

9 return painters.stream()

10 .filter(Painter::isAvailable)

11 .min(Comparator.comparing(painter -> painter.estimateCompensation(sqMeters))));

12 }

13

14 private static Optional<Painter> findCheapest2(double sqMeters, List<Painter> painters) {

15 return Painter.stream(painters).available().cheapest(sqMeters);

16 }

17

18 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

19 return painters.stream() Stream<Painter>

20 .filter(Painter::isAvailable) Stream<Painter>

21 .map(painter -> painter.estimateCompensation(sqMeters)) Stream<Money>

22 .reduce(Money::add) Optional<Money>

23 .orElse(Money.ZERO);

24 }

25

26 public void workTogether(double sqMeters, List<Painter> painters) {

27

28 }

29

30 public void run() {

31 }

32 }

33

Demo > workTogether()

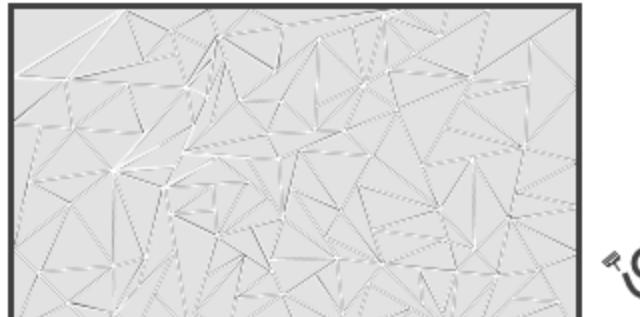
2: Favorites

3: I: Structure

4: Favorites

Ant Build

m Maven



Event Log

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X PaintersStream.java X Main.java X

Project

1: Project

2: Favorites

3: I: Structure

4: 2: Favorites

6

7 public class Demo {

8 @ private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {

9 return painters.stream()

10 .filter(Painter::isAvailable)

11 .min(Comparator.comparing(painter -> painter.estimateCompensation(sqMeters)));

12 }

13

14 private static Optional<Painter> findCheapest2(double sqMeters, List<Painter> painters) {

15 return Painter.stream(painters).available().cheapest(sqMeters);

16 }

17

18 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

19 return painters.stream() Stream<Painter>

20 .filter(Painter::isAvailable) Stream<Painter>

21 .map(painter -> painter.estimateCompensation(sqMeters)) Stream<Money>

22 .reduce(Money::add) Optional<Money>

23 .orElse(Money.ZERO);

24 }

25

26 public void workTogether(double sqMeters, List<Painter> painters) {

27 }

28 }

29

30 public void run() {

31 }

32 }

33

Demo > workTogether()

Ant Build

m Maven



Event Log

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java X Demo.java X PaintersStream.java X Main.java X

1: Project

6

7 public class Demo {

8 @ private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {

9 return painters.stream()

10 .filter(Painter::isAvailable)

11 .min(Comparator.comparing(painter -> painter.estimateCompensation(sqMeters))));

12 }

13

14 private static Optional<Painter> findCheapest2(double sqMeters, List<Painter> painters) {

15 return Painter.stream(painters).available().cheapest(sqMeters);

16 }

17

18 @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {

19 return painters.stream() Stream<Painter>

20 .filter(Painter::isAvailable) Stream<Painter>

21 .map(painter -> painter.estimateCompensation(sqMeters)) Stream<Money>

22 .reduce(Money::add) Optional<Money>

23 .orElse(Money.ZERO);

24 }

25

26 public void workTogether(double sqMeters, List<Painter> painters) {

27

28 }

29

30 public void run() {

31 }

32 }

33

Demo > workTogether()

1: Structure

2: Favorites

Ant Build

m Maven



Event Log

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Painter.java X

2: Demo.java X

3: PaintersStream.java X

4: Main.java X

5: Ant Build

6: m Maven

```
public class Demo {  
    @  
    private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {  
        return painters.stream()  
            .filter(Painter::isAvailable)  
            .min(Comparator.comparing(painter -> painter.estimateCompensation(sqMeters)));  
    }  
  
    private static Optional<Painter> findCheapest2(double sqMeters, List<Painter> painters) {  
        return Painter.stream(painters).available().cheapest(sqMeters);  
    }  
  
    @  
    private static Money getTotalCost(double sqMeters, List<Painter> painters) {  
        return painters.stream() Stream<Painter>  
            .filter(Painter::isAvailable) Stream<Painter>  
            .map(painter -> painter.estimateCompensation(sqMeters)) Stream<Money>  
            .reduce(Money::add) Optional<Money>  
            .orElse(Money.ZERO);  
    }  
  
    public void workTogether(double sqMeters, List<Painter> painters) {  
    }  
  
    public void run() {  
    }  
}
```

2: I: Structure

3: Favorites

Demo > workTogether()

6 TODO Terminal Event Log

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Project

1: Project

14
15
16
17
18
19 @
20
21 ↗
22 ↗
23 ↗
24
25
26
27 public void workTogether(double sqMeters, List<Painter> painters) {
28 Velocity groupVelocity =
29 Painter.stream(painters).available()
30 .map(painter -> painter.estimateVelocity(sqMeters)) Stream<Velocity>
31 .reduce(Velocity::add) Optional<Velocity>
32 .orElse(Velocity.ZERO);
33
34 Painter.stream(painters).available()
35 .forEach(painter -> {
36 double partialSqMeters = sqMeters * painter.estimateVelocity(sqMeters).divideBy(groupVelocity);
37 Money partialCost = painter.estimateCompensation(partialSqMeters);
38 Duration partialTime = painter.estimateTimeToPaint(partialSqMeters);
39 });
40 }
41 }

2: Favorites

3: I: Structure

4: Demo > workTogether() > painter->{...}

5: Event Log

6: TODO Terminal

7: Time cost

8: max{time_i} / sum{cost_i}

Ant Build

m Maven

Pluralsight > src > com > codinghelmet > moreoojava > Demo

Main

Painter.java Demo.java Velocity.java PaintersStream.java Main.java

1: Project

14
15
16
17
18
19 @
20
21 ↗
22 ↗
23 ↗
24
25
26
27 public void workTogether(double sqMeters, List<Painter> painters) {
28 Velocity groupVelocity =
29 Painter.stream(painters).available()
30 .map(painter -> painter.estimateVelocity(sqMeters)) Stream<Velocity>
31 .reduce(Velocity::add) Optional<Velocity>
32 .orElse(Velocity.ZERO);
33
34 Painter.stream(painters).available()
35 .forEach(painter -> {
36 double partialSqMeters = sqMeters * painter.estimateVelocity(sqMeters).divideBy(groupVelocity);
37 Money partialCost = painter.estimateCompensation(partialSqMeters);
38 Duration partialTime = painter.estimateTimeToPaint(partialSqMeters);
39 });
40 }
41 }

2: Favorites

3: I: Structure

4: Event Log

5: TODO Terminal

time cost

time cost

Demo > workTogether() > painter->{...}

Summary



Techniques to avoid using loops

- Loops are where the bugs are hiding
- Loop is the coding infrastructure
- Loop's body implements business



Summary



Using streams instead of loops

- Streams API introduced in Java 8
- Primitive transforms can be chained
- Allow complex business operations
- Allow construction of specialized streams



Summary



Use streams

Use custom streams



Summary



Reducing streams

- Select a single object out of many
- Compose a stream into a single result
- More on composition will follow



Summary



Composite objects

- Many objects implement the same interface
- Substitute them with a single object
- Known as the Composite pattern



Summary



Unified collection processing

- Composition and selection principles apply to collections
- Collection acts as a single object
- Consuming code becomes simple



Summary



Next module:

Untangling Operations from
Structure on Business Data

