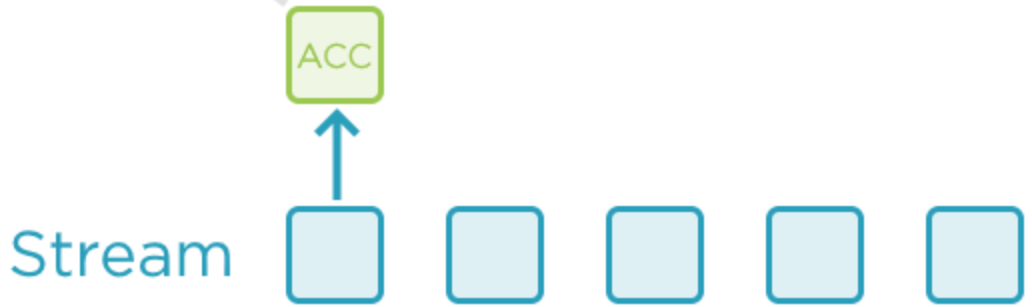
Staying Focused on Domain Logic with Streams

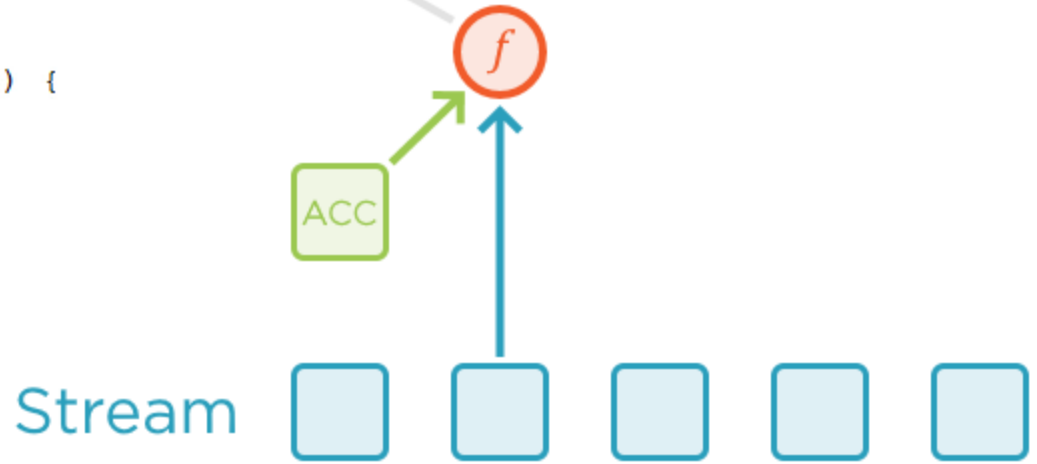
Zoran Horvat
CEO AT CODING HELMET

@zoranh75 http://codinghelmet.com

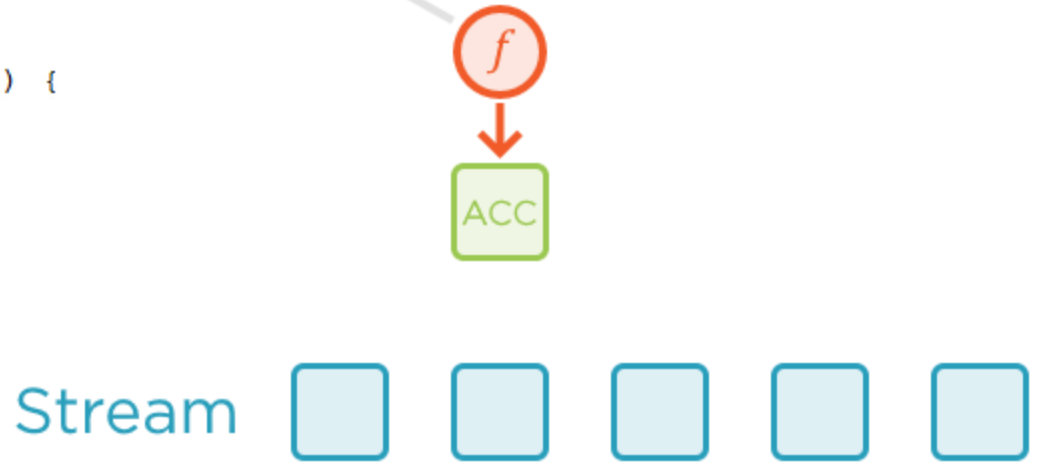
```
5
6   public class Demo {
7       private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
8           return painters.stream()
9               .filter(Painter::isAvailable)
10              .reduce()
11              .get();
12      }
13
14      public void run() {
15      }
16  }
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```



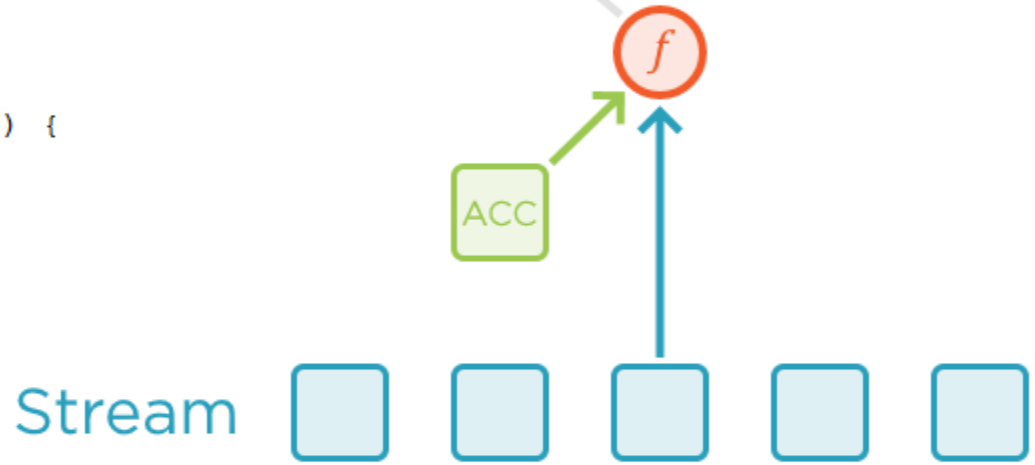
```
5
6   public class Demo {
7       private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
8           return painters.stream()
9               .filter(Painter::isAvailable)
10              .reduce((acc, current) -> )
11              .get();
12      }
13
14      public void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



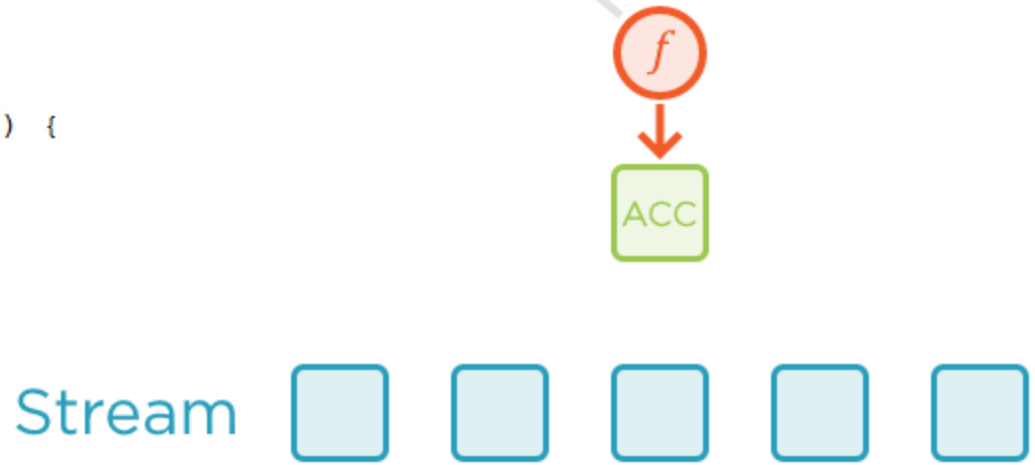
```
5
6   public class Demo {
7       private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
8           return painters.stream()
9               .filter(Painter::isAvailable)
10              .reduce((acc, current) -> )
11              .get();
12      }
13
14      public void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



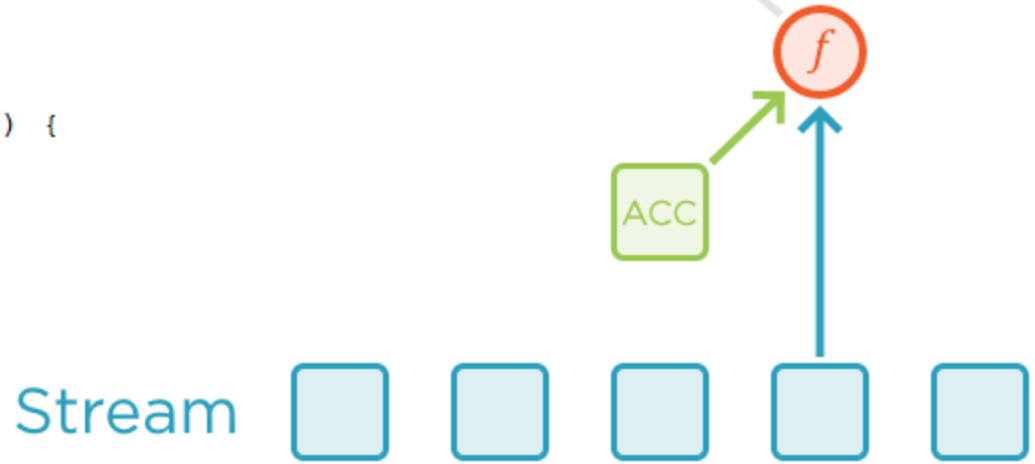
```java

public class Demo {
    @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    public void run() {
    }
}
```



```
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
22
23
24
25
26
27
28
29
30
31
32
```



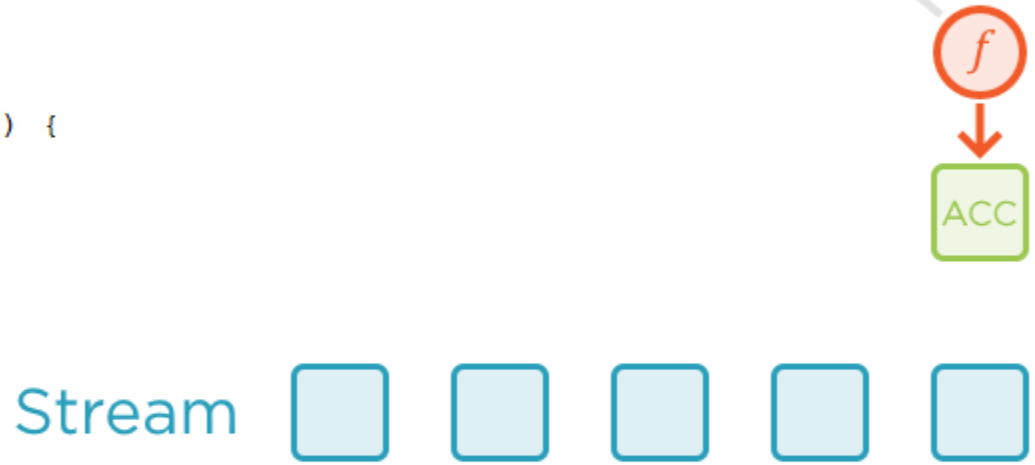
```
5
6   public class Demo {
7       private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
8           return painters.stream()
9               .filter(Painter::isAvailable)
10              .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
11                    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



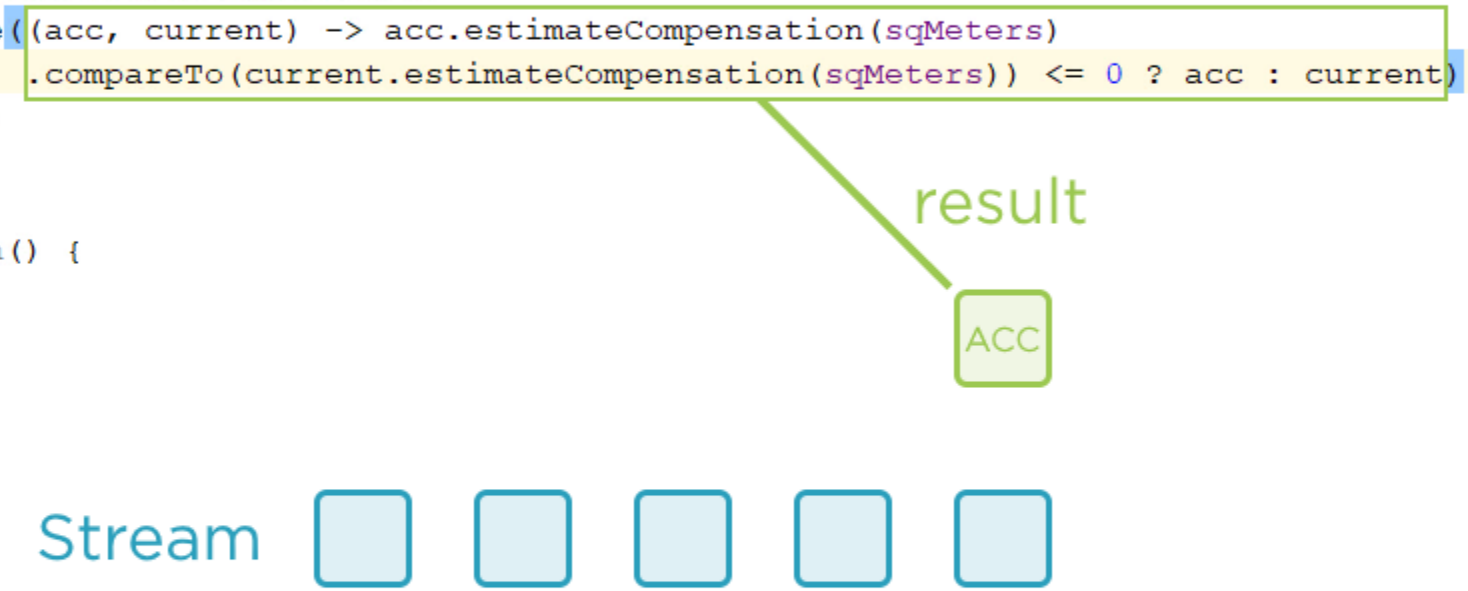
```java

public class Demo {
    @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    public void run() {
    }
}
```

```
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
22
23
24
25
26
27
28
29
30
31
32
```

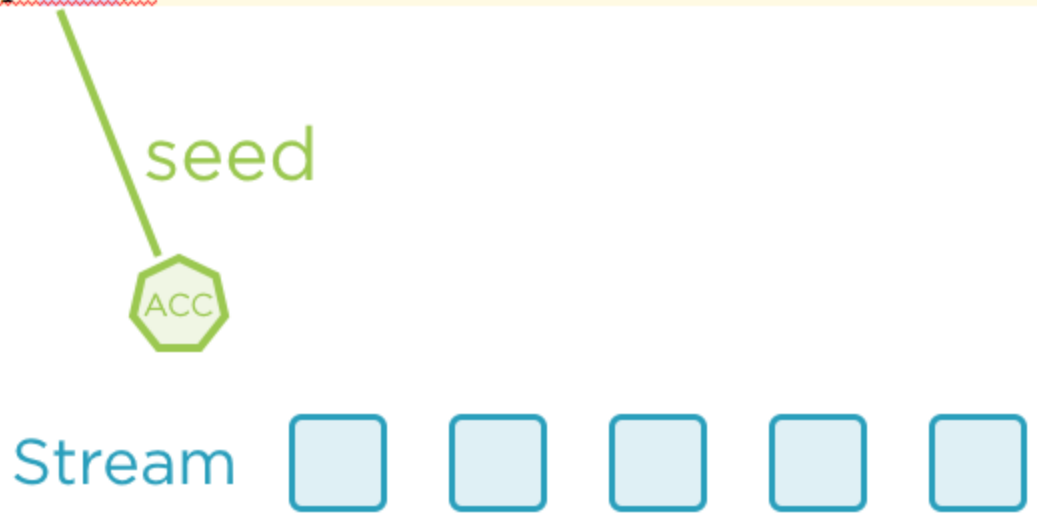
```
5
6   public class Demo {
7       private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
8           return painters.stream()
9               .filter(Painter::isAvailable)
10              .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
11                    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



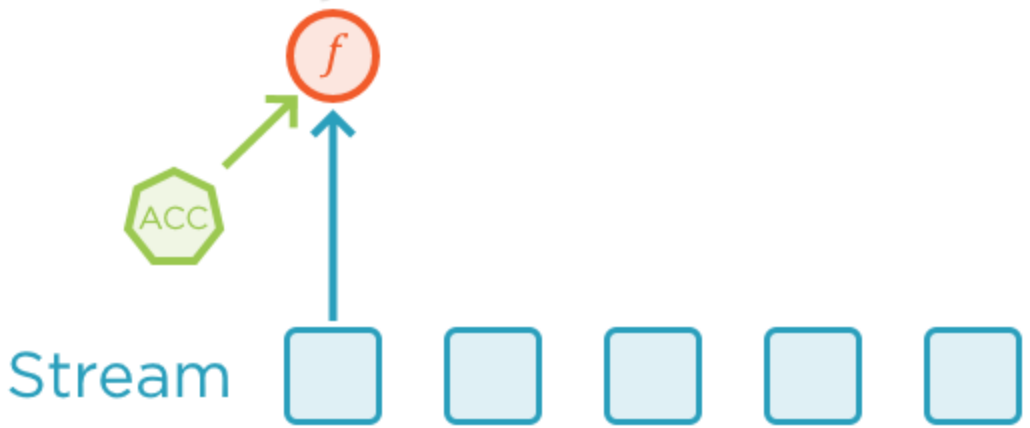
```java

public class Demo {
    @
    private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    public void run() {
    }
}
```



```java

public class Demo {
    private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    private static Money getTotalCost(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce(Money.ZERO)
    }

    public void run() {
    }
}
```



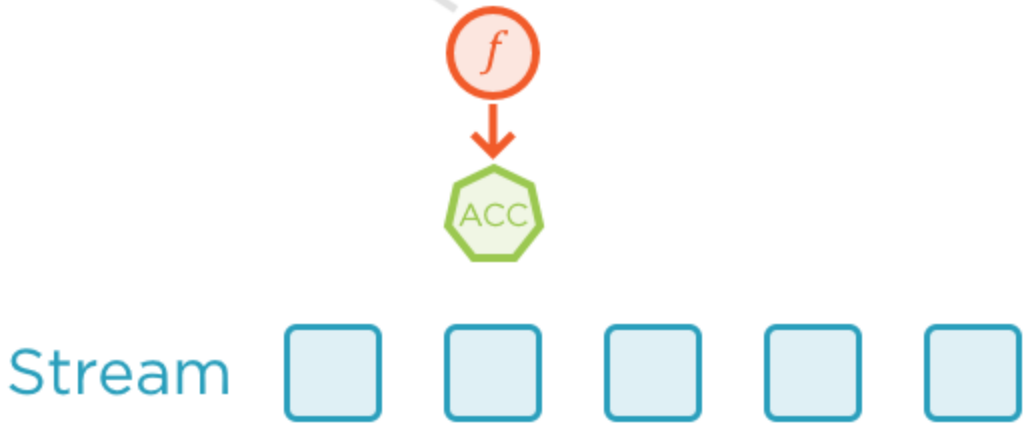
```
5
6   public class Demo {
7       private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
8           return painters.stream()
9               .filter(Painter::isAvailable)
10              .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
11                      .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
29
30
31
32
```



```java

public class Demo {
    private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    private static Money getTotalCost(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce(Money.ZERO, (acc, painter) -> )
    }

    public void run() {
    }
}
```

```java

public class Demo {
    private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    private static Money getTotalCost(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce(Money.ZERO, (acc, painter) -> )
    }

    public void run() {
    }
}
```

```java

public class Demo {
    private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    private static Money getTotalCost(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce(Money.ZERO, (acc, painter) -> )
    }

    public void run() {
    }
}
```



```java

public class Demo {
    private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    private static Money getTotalCost(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce(Money.ZERO, (acc, painter) -> )
    }

    public void run() {
    }
}
```

```java
public class Demo {
    private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    private static Money getTotalCost(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce(Money.ZERO, (acc, painter) -> )
    }

    public void run() {
    }
}
```

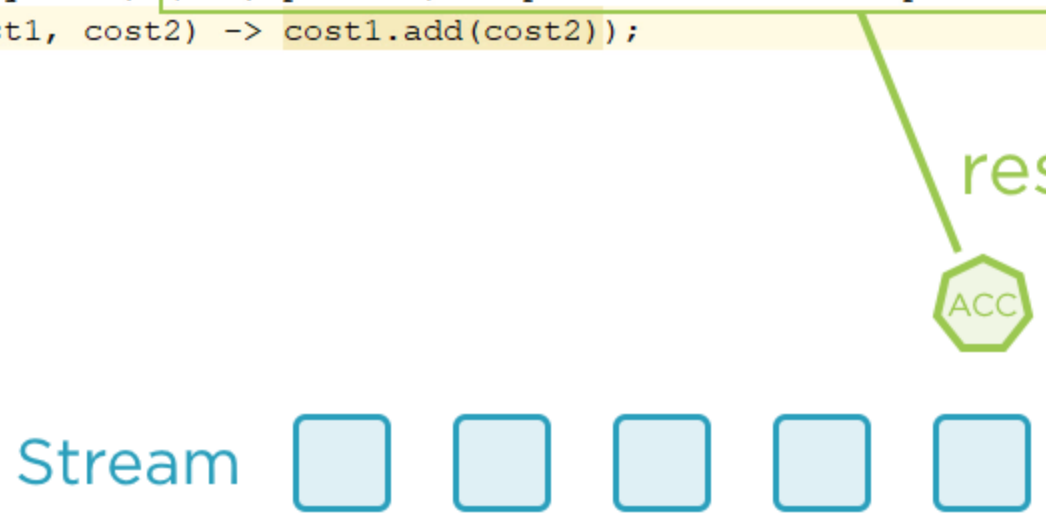
```java

public class Demo {
    private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    private static Money getTotalCost(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce(Money.ZERO, (acc, painter) -> )
    }

    public void run() {
    }
}
```

```
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
29
30
31
32
```

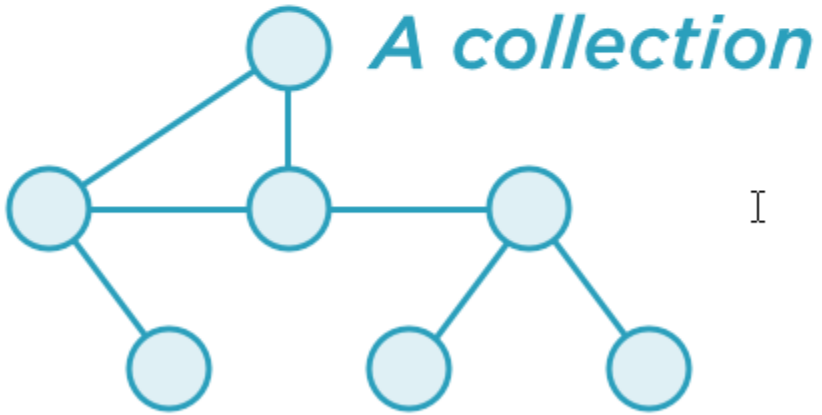
```java

public class Demo {
    private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    private static Money getTotalCost(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce(Money.ZERO, (acc, painter) -> )
    }

    public void run() {
    }
}
```

```java

public class Demo {
    private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    private static Money getTotalCost(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce(Money.ZERO, (acc, painter) -> )
    }

    public void run() {
    }
}
```

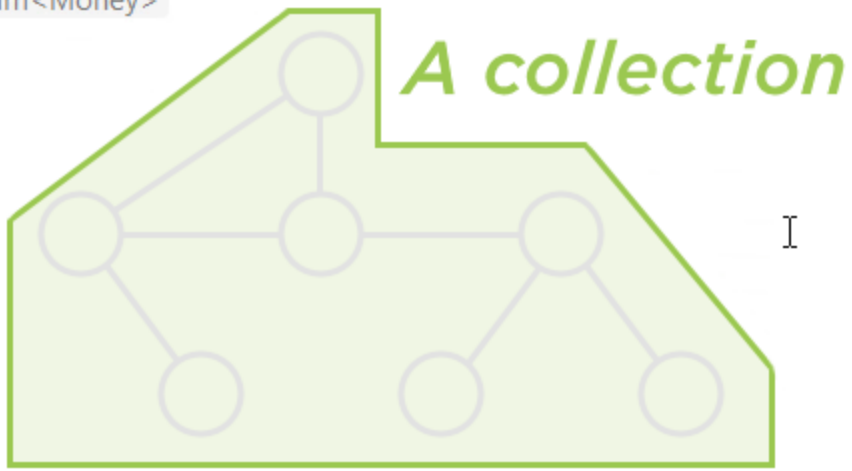
```java

public class Demo {
    @ private static Painter findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce((acc, current) -> acc.estimateCompensation(sqMeters)
                    .compareTo(current.estimateCompensation(sqMeters)) <= 0 ? acc : current)
            .get();
    }

    @ private static Money getTotalCost(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .reduce(Money.ZERO, (acc, painter) -> painter.estimateCompensation(sqMeters).add(acc),
                    (cost1, cost2) -> cost1.add(cost2));
    }

    public void run() {
    }
}
```

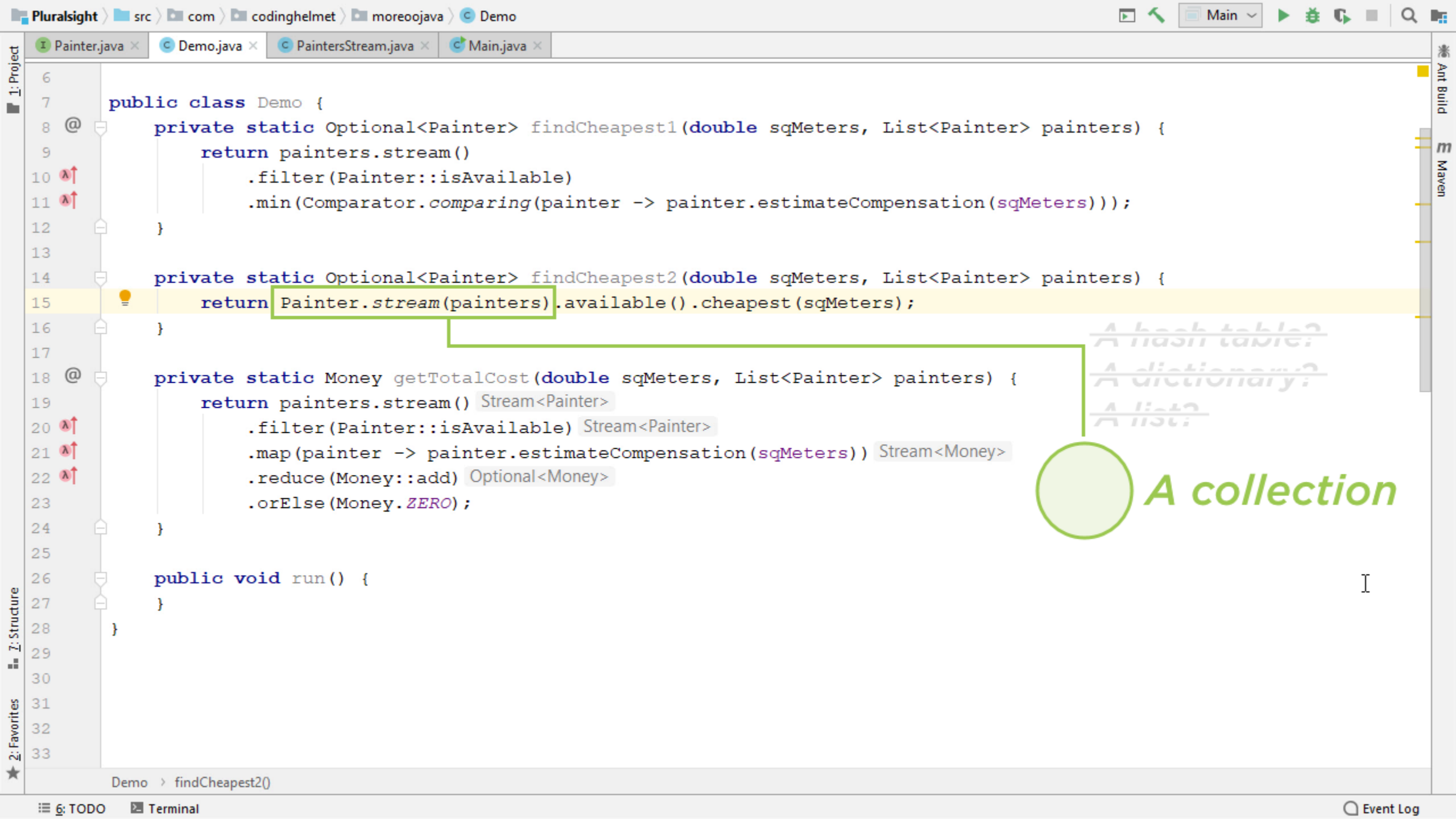


```java
public class Demo {
    private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {
        return painters.stream()
            .filter(Painter::isAvailable)
            .min(Comparator.comparing(painter -> painter.estimateCompensation(sqMeters)));
    }

    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 run() {
    }
}
```



```java

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 run() {
    }
}
```



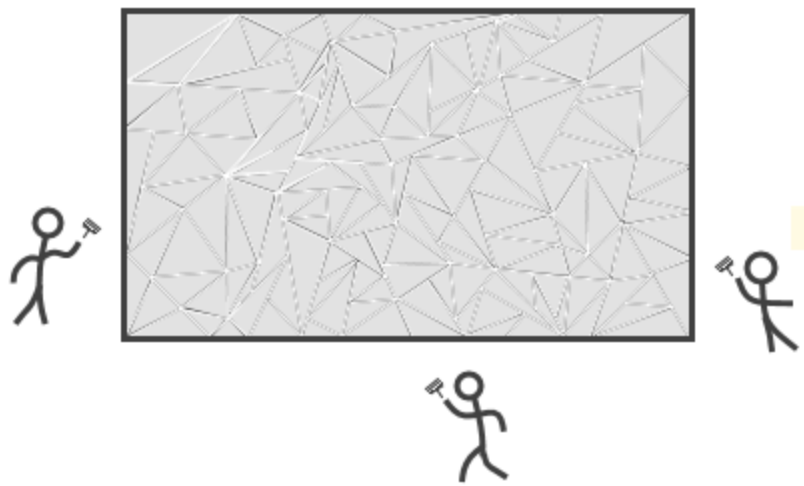

Pluralsight src com codinghelmet moreoojava Demo
Main
Painter.java
Demo.java
PaintersStream.java
Main.java
1: Project
Ant Build
Maven
public class Demo {
private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {
return painters.stream()
.filter(Painter::isAvailable)
.min(Comparator.comparing(painter -> 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 run() {
}
}
A hash table?
A dictionary?
A list?
A collection
Demo › findCheapest2()
7: Structure
2: Favorites
6: TODO
Terminal
Event Log

```
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
```

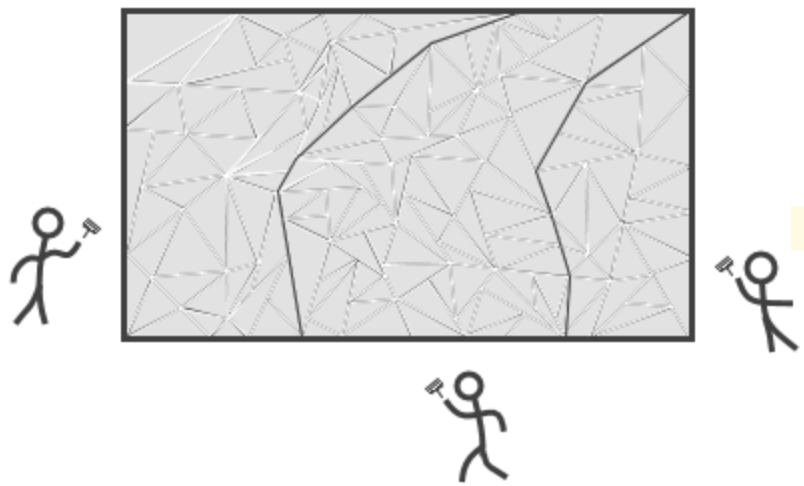
A stream ✓

A collection

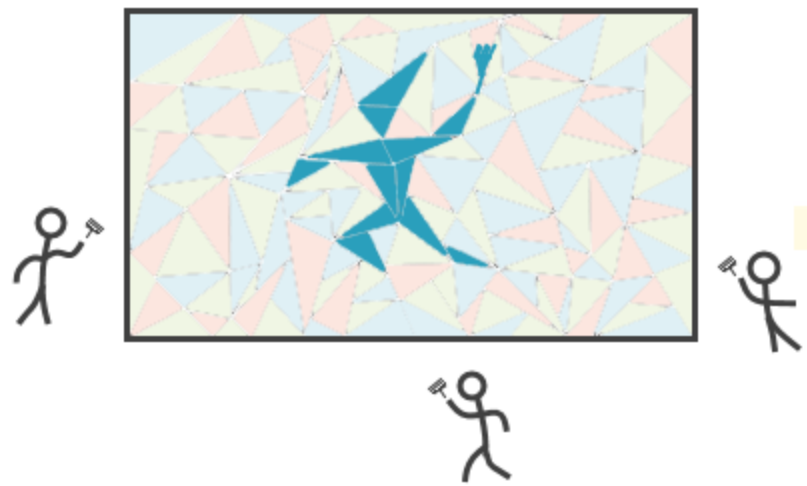
```
6
7   public class Demo {
8       private static Optional<Painter> findCheapest1(double sqMeters, List<Painter> painters) {
9           return painters.stream()
10              .filter(Painter::isAvailable)
11              .min(Comparator.comparing(painter -> painter.estimateCompensation(sqMeters)));
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
```



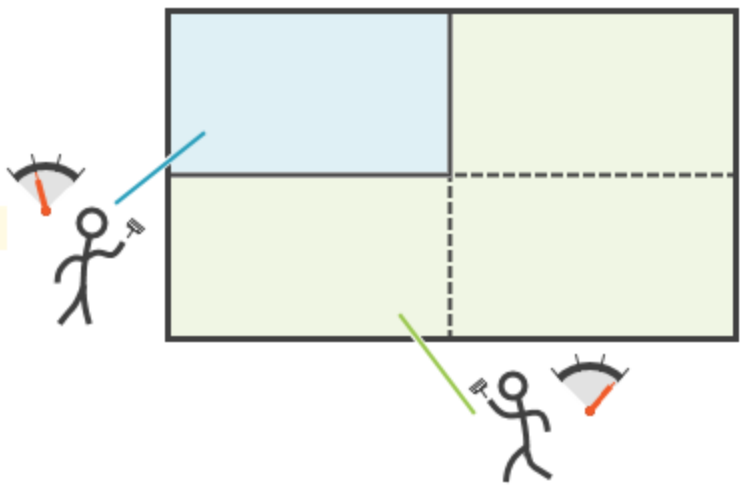
```java

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() {
    }
}

```



```
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
```



```
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
```




```
14
15
16
17
18
19  @
20
21  λ↑
22  λ↑
23  λ↑
24
25
26
27      public void workTogether(double sqMeters, List<Painter> painters) {
28          Velocity groupVelocity =
29              Painter.stream(painters).available()  PaintersStream
30  λ↑              .map(painter -> painter.estimateVelocity(sqMeters))  Stream<Velocity>
31  λ↑              .reduce(Velocity::add)  Optional<Velocity>
32                  .orElse(Velocity.ZERO);
33
34          Painter.stream(painters).available()
35  λ↑              .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
```


```
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()  PaintersStream
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
```

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