CS5800 – Advanced Software Engineering Professor Nima Davarpanah Homework 4 – Structural Design Pattern Devaansh Mann

GitHub Link: https://github.com/DevaanshMann/CS5800-HW4

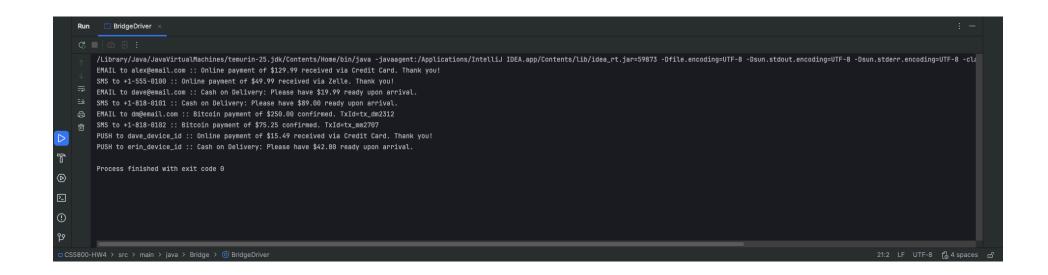


```
1 package Bridge;
 2
 3 import java.util.*;
 5 abstract class PaymentNotification {
 6
       protected final NotificationChannel channel;
 7
       protected final String customer;
 8
       protected final double amount;
 9
10
       protected PaymentNotification(
   NotificationChannel channel, String customer,
   double amount) {
11
           this.channel = channel;
12
           this.customer = customer;
13
           this.amount = amount;
14
       }
15
       public abstract void notifyCustomer();
16 }
17
18 interface NotificationChannel {
       void send(String to, String message);
19
20 }
21
22 class EmailChannel implements NotificationChannel {
23
       @Override
24
       public void send(String to, String message) {
           System.out.println("EMAIL to " + to +
25
     :: " + message);
26
       }
27 }
28
29 class SmsChannel implements NotificationChannel {
30
       @Override
       public void send(String to, String message) {
31
           System.out.println("SMS to " + to + " :: "
32
    + message);
33
       }
34 }
35
36 class PushChannel implements NotificationChannel {
37
       @Override
```

```
public void send(String to, String message) {
38
39
           System.out.println("PUSH to " + to + " :: "
    + message);
40
       }
41 }
42
43 class OnlinePaymentNotification extends
   PaymentNotification {
       private final String method;
44
45
       public OnlinePaymentNotification(
   NotificationChannel c, String customer, double
   amount, String method) {
46
           super(c, customer, amount);
           this.method = method;
47
48
       }
49
       00verride
50
       public void notifyCustomer() {
           channel.send(customer, String.format("
51
   Online payment of $%.2f received via %s. Thank you
   !", amount, method));
52
       }
53 }
54
55 class CashOnDeliveryPayment extends
   PaymentNotification {
       public CashOnDeliveryPayment(
56
   NotificationChannel c, String customer, double
   amount) {
57
           super(c, customer, amount);
58
       }
59
       @Override
       public void notifyCustomer() {
60
           channel.send(customer, String.format("Cash
61
   on Delivery: Please have $%.2f ready upon arrival."
   , amount));
62
       }
63 }
64
65 class BitcoinPayment extends PaymentNotification {
       private final String txId;
66
67
       public BitcoinPayment(NotificationChannel c,
```

```
67 String customer, double amount, String txId) {
           super(c, customer, amount);
68
69
           this.txId = txId;
70
       }
71
       @Override
72
       public void notifyCustomer() {
73
           channel.send(customer, String.format("
   Bitcoin payment of $%.2f confirmed. TxId=%s",
   amount, txId));
74
       }
75 }
76
77
78
```

```
1 package Bridge;
 2
 3 public class BridgeDriver {
       public static void main(String[] args) {
 4
 5
           NotificationChannel email = new
   EmailChannel();
           NotificationChannel sms = new SmsChannel
 6
   ();
 7
           NotificationChannel push = new PushChannel
   ();
 8
           new OnlinePaymentNotification(email, "alex@
   email.com", 129.99, "Credit Card").notifyCustomer
   ();
10
           new OnlinePaymentNotification(sms, "+1-555-
   0100", 49.99, "Zelle").notifyCustomer();
11
12
           new CashOnDeliveryPayment(email, "dave@
   email.com", 19.99).notifyCustomer();
13
           new CashOnDeliveryPayment(sms, "+1-818-0101
   ", 89.00).notifyCustomer();
14
15
           new BitcoinPayment(email, "dm@email.com",
   250.00, "tx_dm2312").notifyCustomer();
           new BitcoinPayment(sms, "+1-818-0102", 75.
16
   25, "tx_mm2707").notifyCustomer();
17
18
           new OnlinePaymentNotification(push, "
   dave_device_id", 15.49, "Credit Card").
   notifyCustomer();
19
           new CashOnDeliveryPayment(push, "
   erin_device_id", 42.00).notifyCustomer();
20
       }
21 }
```





```
1 package Decorator;
 2
 3 import java.util.*;
5 interface FoodItem {
       String description();
 6
 7
       double cost();
8 }
 9
10 class Burger implements FoodItem {
11
       @Override
       public String description() {
12
           return "Burger";
13
14
15
       @Override
16
       public double cost() {
17
           return 5.00;
18
       }
19 }
20
21 class Fries implements FoodItem {
22
       @Override
       public String description() {
23
24
           return "Fries";
25
       }
26
       @Override
27
       public double cost() {
28
           return 2.50;
29
       }
30 }
31
32 class HotDog implements FoodItem {
33
       @Override
       public String description() {
34
35
           return "Hot Dog";
36
       }
37
38
       @Override
       public double cost() {
39
40
           return 3.50;
       }
41
```

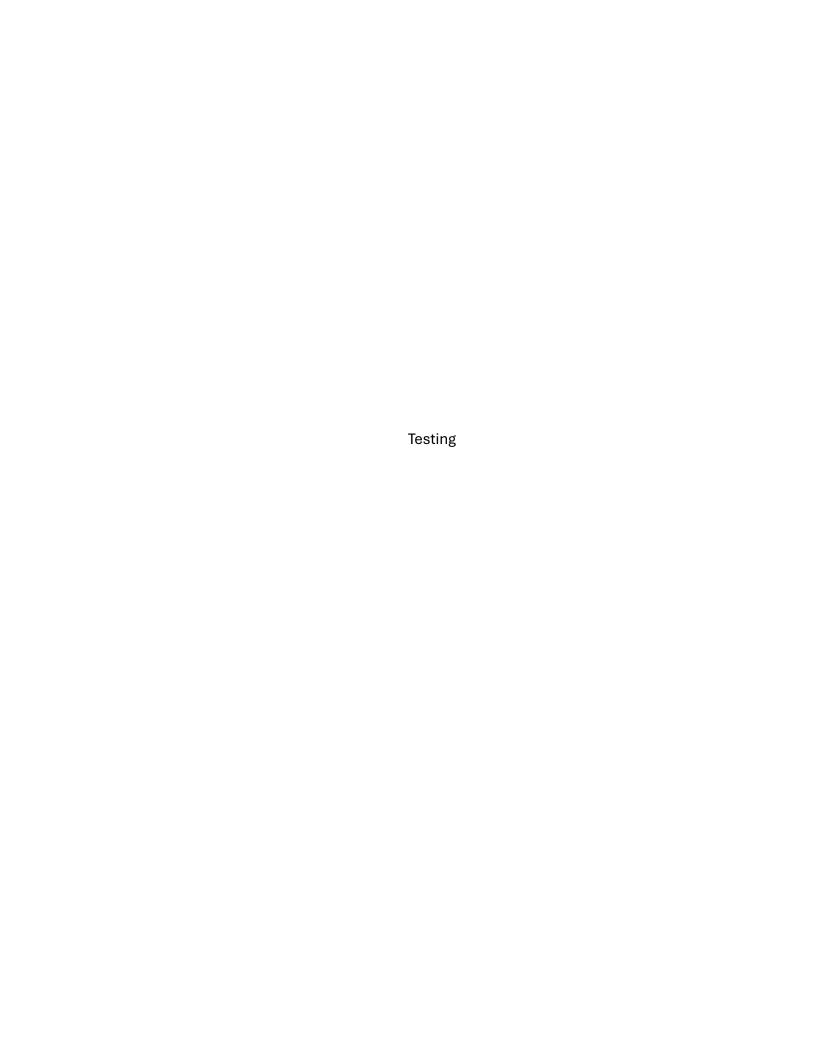
```
42 }
43
44 abstract class ToppingDecorator implements FoodItem
45
       protected final FoodItem base;
46
       protected ToppingDecorator(FoodItem base) {
47
           this.base = base;
48
       }
49 }
50
51 class Ketchup extends ToppingDecorator {
       public Ketchup(FoodItem base) {
52
           super(base);
53
54
       }
55
56
       @Override
57
       public String description() {
           return base.description() + " + Ketchup";
58
59
       }
60
61
       @Override
62
       public double cost() {
           return base.cost() + 0.20;
63
64
       }
65 }
66
67 class Cheese extends ToppingDecorator {
       public Cheese(FoodItem base) {
68
           super(base);
69
70
       }
71
72
       @Override
       public String description() {
73
           return base.description() + " + Cheese";
74
75
       }
76
77
       @Override
       public double cost() {
78
79
           return base.cost() + 0.50;
80
       }
81 }
```

```
82
 83 class Onions extends ToppingDecorator {
 84
        public Onions(FoodItem base) {
 85
            super(base);
 86
        }
 87
 88
        @Override
 89
        public String description() {
            return base.description() + " + Onions";
 90
 91
        }
 92
 93
        @Override
 94
        public double cost() {
 95
            return base.cost() + 0.30;
 96
        }
 97 }
 98
 99 class Order {
        private final List<FoodItem> items = new
100
    ArrayList<>();
101
102
        public void add(FoodItem item) {
            items.add(item);
103
        }
104
105
106
        public double subtotal() {
            return items.stream().mapToDouble(FoodItem
107
    ::cost).sum();
108
        }
109
110
        public void printItems() {
            for (FoodItem i : items) {
111
                System.out.printf(" - %-40s $%.2f%n",
112
    i.description(), i.cost());
113
114
        }
115 }
116
117 enum LoyaltyTier {
        NONE(0.00), SILVER(0.05), GOLD(0.10), PLATINUM
118
    (0.20);
```

```
File - /Users/devaanshmann/Desktop/CS5800-HW4/src/main/java/Decorator/Decorator.java
119
         final double discountRate;
         LoyaltyTier(double r) {
120
121
              this.discountRate = r;
122
         }
123 }
124
125 class LoyaltyStatus {
126
         private final LoyaltyTier tier;
127
         public LoyaltyStatus(LoyaltyTier tier) {
128
              this.tier = tier;
129
         }
         public double applyDiscount(double amount) {
130
131
              return amount * (1.0 - tier.discountRate);
132
         }
         public LoyaltyTier tier() {
133
134
              return tier;
         }
135
136 }
137
138
139
```

```
1 package Decorator;
 2
 3 public class DecoratorDriver {
       public static void main(String[] args) {
 5
           FoodItem item1 = new Cheese(new Ketchup(new
 6
    Onions(new Burger()));
 7
           FoodItem item2 = new Onions(new Cheese(new
   HotDog());
           FoodItem item3 = new Ketchup(new Fries());
 8
 9
10
           Order order = new Order();
11
           order.add(item1);
12
           order.add(item2);
13
           order.add(item3);
14
15
                                          DECORATOR
           System.out.println("
   DEMO");
16
           order.printItems();
17
           double subtotal = order.subtotal();
           System.out.printf("Subtotal: $%.2f%n",
18
   subtotal);
19
20
           LoyaltyStatus status = new LoyaltyStatus(
   LoyaltyTier.PLATINUM);
21
           double totalAfterDiscount = status.
   applyDiscount(subtotal);
22
           System.out.printf("Loyalty Tier: %s (%.0f
   %% off)%n", status.tier(), status.tier().
   discountRate * 100);
23
           System.out.printf("Total after discount: $
  %.2f%n", totalAfterDiscount);
24
       }
25 }
```



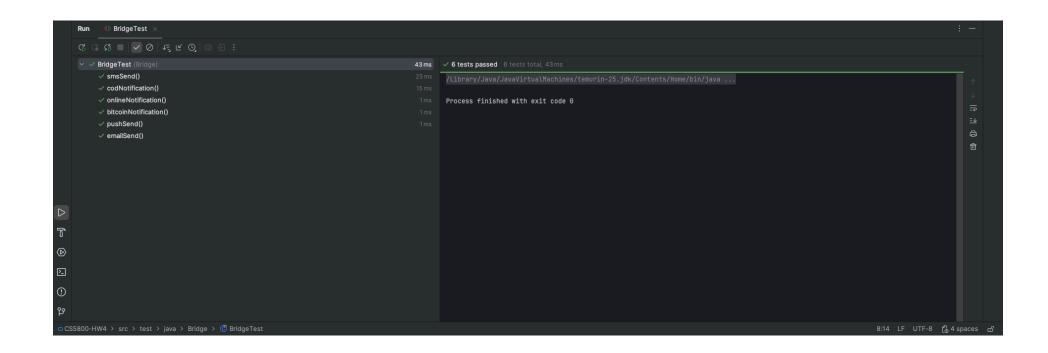


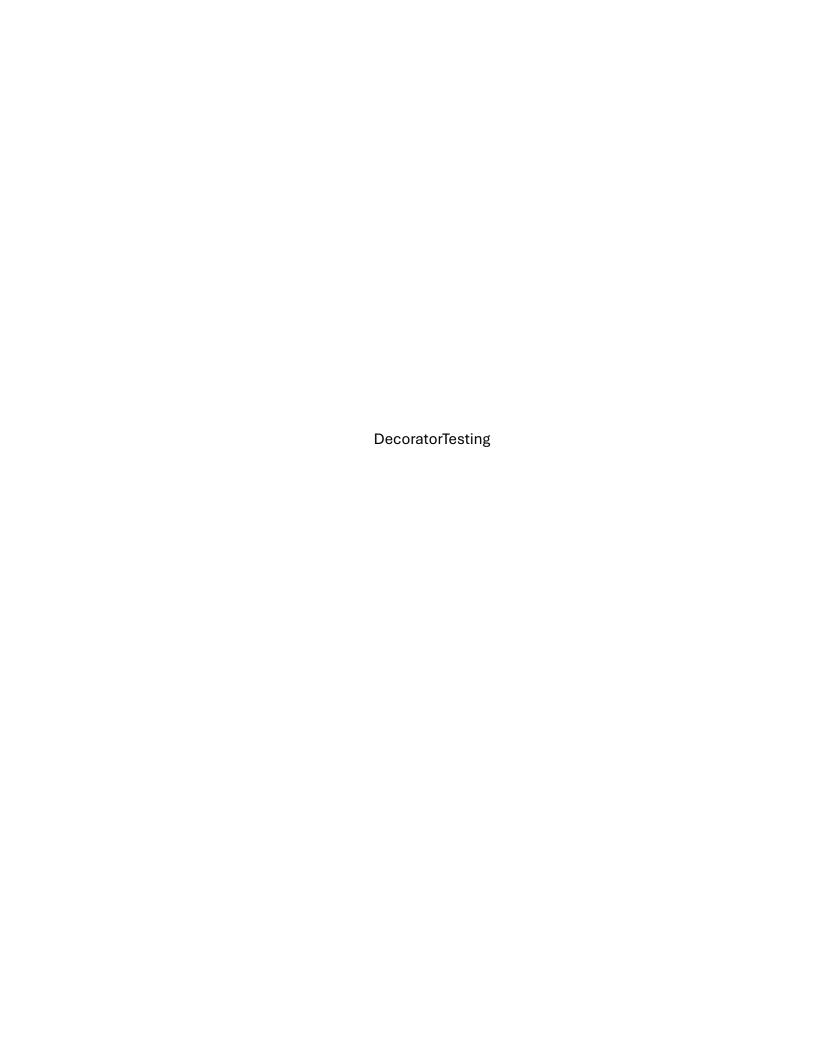


```
1 package Bridge;
 2
 3 import org.junit.jupiter.api.*;
 4 import java.io.ByteArrayOutputStream;
 5 import java.io.PrintStream;
 6 import static org.junit.jupiter.api.Assertions.*;
 7
8 public class BridgeTest {
 9
10
       private final PrintStream originalOut = System.
   out;
11
       private ByteArrayOutputStream capture;
12
       @BeforeEach
13
14
       void setUp() {
15
           capture = new ByteArrayOutputStream();
           System.setOut(new PrintStream(capture));
16
17
       }
18
19
       @AfterEach
20
       void tearDown() {
           System.setOut(originalOut);
21
22
       }
23
24
       @Test
25
       void emailSend() {
           NotificationChannel ch = new EmailChannel
26
   ():
27
           ch.send("user@email.com", "pass");
28
           String out = capture.toString();
29
           assertTrue(out.contains("EMAIL to user@
   email.com :: pass"));
30
       }
31
32
       @Test
33
       void smsSend() {
34
           NotificationChannel ch = new SmsChannel();
           ch.send("+1-555-0101", "otp:1234");
35
           String out = capture.toString();
36
           assertTrue(out.contains("SMS to +1-555-0101
37
    :: otp:1234"));
```

```
38
39
40
       @Test
41
       void pushSend() {
42
           NotificationChannel ch = new PushChannel();
           ch.send("device_abc", "received");
43
44
           String out = capture.toString();
45
           assertTrue(out.contains("PUSH to device_abc
    :: received"));
46
       }
47
48
       @Test
49
       void onlineNotification() {
50
           NotificationChannel email = new
   EmailChannel();
51
           PaymentNotification n = new
   OnlinePaymentNotification(email, "alex@email.com",
   129.99, "Credit Card");
52
           n.notifyCustomer();
53
           String out = capture.toString();
           assertTrue(out.contains("EMAIL to alex@
54
   email.com"));
55
           assertTrue(out.contains("Online payment of
   $129.99 received via Credit Card. Thank you!"));
56
       }
57
58
       @Test
59
       void codNotification() {
60
           NotificationChannel sms = new SmsChannel();
61
           PaymentNotification n = new
   CashOnDeliveryPayment(sms, "+1-555-0101", 89.00);
62
           n.notifyCustomer();
63
           String out = capture.toString();
           assertTrue(out.contains("SMS to +1-555-0101
64
   "));
           assertTrue(out.contains("Cash on Delivery:
65
   Please have $89.00 ready upon arrival."));
       }
66
67
68
       @Test
69
       void bitcoinNotification() {
```

```
File - /Users/devaanshmann/Desktop/CS5800-HW4/src/test/java/Bridge/BridgeTest.java
              NotificationChannel push = new PushChannel
 70
     ();
              PaymentNotification n = new BitcoinPayment
 71
     (push, "dev_device", 15.49, "tx_abc");
              n.notifyCustomer();
 72
              String out = capture.toString();
 73
              assertTrue(out.contains("PUSH to
 74
     dev_device"));
              assertTrue(out.contains("Bitcoin payment
 75
     of $15.49 confirmed. TxId=tx_abc"));
 76
         }
 77 }
 78
```



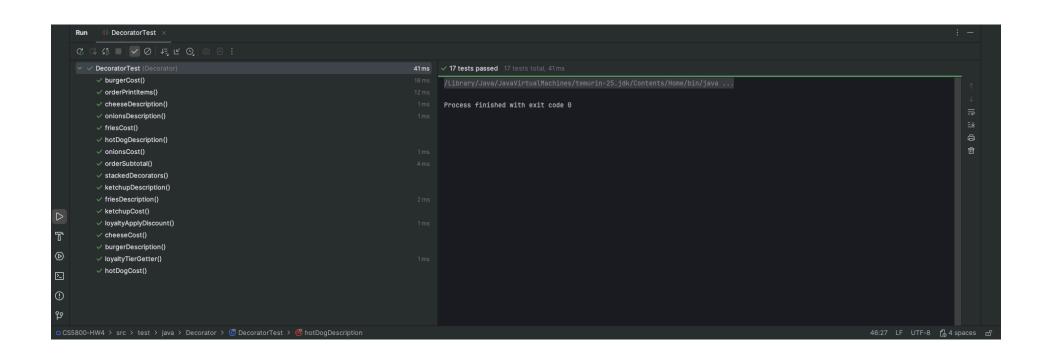


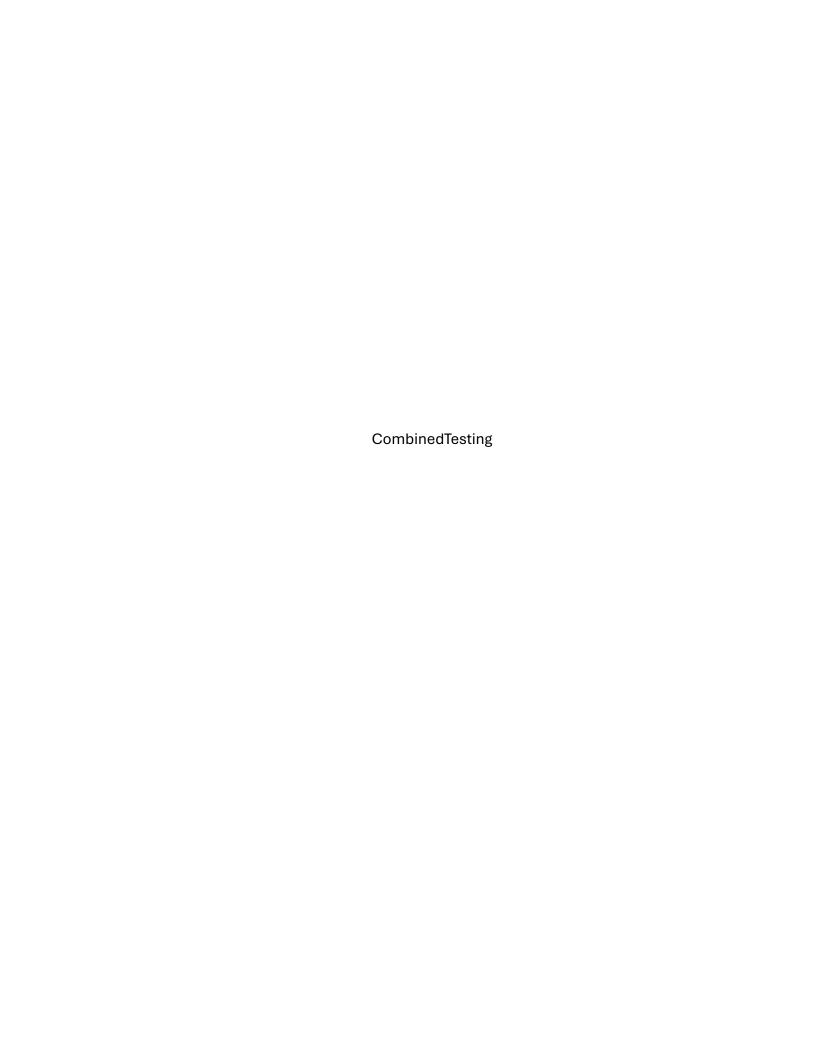
```
1 package Decorator;
 2
 3 import org.junit.jupiter.api.*;
 4 import java.io.ByteArrayOutputStream;
 5 import java.io.PrintStream;
6 import static org.junit.jupiter.api.Assertions.*;
 7
 8 public class DecoratorTest {
 9
10
       private final PrintStream originalOut = System.
   out;
11
       private ByteArrayOutputStream capture;
12
13
       @BeforeEach
14
       void setUp() {
15
           capture = new ByteArrayOutputStream();
           System.setOut(new PrintStream(capture));
16
17
       }
18
19
       @AfterEach
20
       void tearDown() {
           System.setOut(originalOut);
21
22
       }
23
24
       @Test void burgerDescription() {
25
           FoodItem b = new Burger();
           assertEquals("Burger", b.description());
26
       }
27
28
29
       @Test void burgerCost() {
30
           FoodItem b = new Burger();
           assertEquals(5.00, b.cost(), 1e-2);
31
32
       }
33
34
       @Test void friesDescription() {
35
           FoodItem f = new Fries();
           assertEquals("Fries", f.description());
36
37
       }
38
39
       @Test void friesCost() {
40
           FoodItem f = new Fries();
```

```
41
           assertEquals(2.50, f.cost(), 1e-2);
42
       }
43
44
       @Test void hotDogDescription() {
           FoodItem h = new HotDog();
45
           assertEquals("Hot Dog", h.description());
46
       }
47
48
49
       @Test void hotDogCost() {
50
           FoodItem h = new HotDog();
           assertEquals(3.50, h.cost(), 1e-2);
51
52
       }
53
54
       @Test void ketchupDescription() {
55
           FoodItem item = new Ketchup(new Burger());
           assertEquals("Burger + Ketchup", item.
56
   description());
57
       }
58
59
       @Test void ketchupCost() {
60
           FoodItem item = new Ketchup(new Burger());
61
           assertEquals(5.00 + 0.20, item.cost(), 1e-2
   );
62
       }
63
       @Test void cheeseDescription() {
64
           FoodItem item = new Cheese(new Fries());
65
           assertEquals("Fries + Cheese", item.
66
   description());
67
       }
68
69
       @Test void cheeseCost() {
70
           FoodItem item = new Cheese(new Fries());
           assertEquals(2.50 + 0.50, item.cost(), 1e-2
71
   );
72
       }
73
74
       @Test void onionsDescription() {
75
           FoodItem item = new Onions(new HotDog());
           assertEquals("Hot Dog + Onions", item.
76
   description());
```

```
77
 78
 79
        @Test void onionsCost() {
 80
            FoodItem item = new Onions(new HotDog());
            assertEquals(3.50 + 0.30, item.cost(), 1e-
 81
    2);
        }
 82
 83
 84
        @Test void orderSubtotal() {
 85
            Order order = new Order();
            order.add(new Cheese(new Ketchup(new
 86
    Burger()))); // 5.70
            order.add(new Onions(new Cheese(new HotDog
 87
    ())));
            // 4.30
            order.add(new Ketchup(new Fries
 88
    ()));
                        // 2.70
            assertEquals(12.70, order.subtotal(), 1e-2
 89
    );
        }
 90
 91
 92
        @Test void orderPrintItems() {
 93
            Order order = new Order();
            FoodItem item = new Cheese(new Ketchup(new
 94
     Burger())); // "Burger + Ketchup + Cheese"
 95
            order.add(item);
 96
            order.printItems();
            String out = capture.toString();
 97
 98
            assertTrue(out.contains("Burger + Ketchup
     + Cheese"));
 99
            assertTrue(out.contains("$5.70"));
100
        }
101
102
        @Test void loyaltyApplyDiscount() {
103
            LoyaltyStatus status = new LoyaltyStatus(
    LoyaltyTier.GOLD);
104
            assertEquals(100.0 * 0.90, status.
    applyDiscount(100.0), 1e-2);
105
        }
106
107
        @Test void loyaltyTierGetter() {
108
            LoyaltyStatus status = new LoyaltyStatus(
```

```
108 LoyaltyTier.SILVER);
            assertEquals(LoyaltyTier.SILVER, status.
109
    tier());
        }
110
111
        @Test void stackedDecorators() {
112
113
            FoodItem item = new Onions(new Cheese(new
    Ketchup(new Burger())));
            assertEquals(6.00, item.cost(), 1e-2);
114
            assertEquals("Burger + Ketchup + Cheese +
115
    Onions", item.description());
116
        }
117 }
118
119
```





```
1 package CombinedTest;
2
3 import org.junit.platform.suite.api.SelectPackages;
4 import org.junit.platform.suite.api.Suite;
5
6 @Suite
7 @SelectPackages({"Bridge", "Decorator"})
8 public class combinedTest {
9 }
10
```

